The Ribosome Builder:

A Software Project to Simulate the Ribosome

by

William A. Knight

B.Sc. California Polytechnic State University, United States, 1990

presented in partial fulfillment of the requirements

for the degree of

Doctorate

The University of Montana

April 2005

Approved by:

_____

Chairperson

_____

Dean, Graduate School

_____

Date

Knight, William A. Ph.D., April 2005 Biochemistry

The Ribosome Builder: A Software Project to Simulate the Ribosome

Director: J. Stephen Lodmell

The Ribosome Builder is a software program that is designed to efficiently represent and simulate large atomic-resolution structural models of macromolecules such as the ribosome. The representation includes a simplified molecular dynamics forcefield that maintains fundamental properties such as bond lengths, bond angles and steric exclusions. A new method, called the 'shadow-point' method, was developed to represent the covalent bond structure in a way that is robust in the presence of strong external forces. This was done to facilitate applications of Steered Molecular Dynamics (SMD). A high-level scripting engine is also provided for defining the SMD simulations. The scripting engine uses the Lua programming language and includes a comprehensive Application Programming Interface (API). The scripting API enables complex and flexible forces and torques to be defined by Lua scripts and then applied to dynamic models in the forcefield. This capability enabled the creation of SMD simulations of long-timescale activity such as the folding of an RNA tetraloop, docking of mRNA on the ribosome, and translocation of the mRNA during the elongation cycle. The project also introduces a technique called 'schematic simulation'. This technique was proposed as an additional means for simulating long-timescale activity. A schematic simulation is a scripted sequence of events intended to represent a complex macromolecular process. This kind of simulation employs static and dynamic models, descriptive text, graphical annotations, changing viewpoint, scripted movements and playback of recorded molecular dynamics trajectories. A schematic simulation of a complete cycle of elongation on the ribosome was created, and includes all of the principal molecular components at an atomic level of detail. In addition to the core functionality, a comprehensive graphical user interface is provided, and facilitates the construction of multi-component models of macromolecular systems. The software was developed in a portable, modular and documented manner using the C++, Lua and Perl programming languages. The program has been released as an open source project under the GNU public license and is available from SourceForge at http://rbuilder.sourceforge.net.

Acknowledgements

My odyssey of the ribosome began when I first met Walt Hill in 1996 when I took a class of his at the U of M. Walt's initial encouragement for me to pursue a graduate degree and his continuing friendship and support over the years are the primary cause of this work.

During a period when Walt disappeared and went to Las Vegas, his role as mentor was generously taken over by Steve Lodmell, whose good humor, professional ability and unflagging energy sustained both myself and those others still remaining in Walt's lab. Steve's friendship, support, advice and encouragement have been essential factors in the completion of this work.

In addition to these two advisors, I would like to thank the rest of those on my committee: Scott Samuels, Barb Wright, Alden Wright and Michele McGuirl. Their patience, consideration and time have been strongly appreciated.

I would also like to acknowledge Greg Muth, Scott Hennelly, Jing Yuan, Bill Bowen and Marty Rice, whose friendship both in the lab and out have created fond and lasting memories. I regret that I have not spent more time in their company.

I would like to express my appreciation for the faculty and administration of the University of Montana. I am originally from California and moved to Montana 10 years ago. I have been impressed in a general way by the friendly and professional character of those that I have met and worked with during my time here as a student at the U of M.

Although I have not known them personally, I would like to thank the following people for the generous contributions of their creative abilities to the general public. The results of their work have helped to produce the Ribosome Builder software: Richard Stallman, for keeping software free, Roger Sayle for creating RasMol, Bjarne Stroustrup for creating C++, Roberto Ierusalimschy, Luiz Henrique de Figueiredo and Waldemar Celes for creating Lua, Trolltech for creating the Qt library, and the founders of SourceForge for creating a public open source software repository.

Another individual who has contributed important support for the program is Drew Kearns, in his design of the FPSocket library, and his very generous contribution of his high-speed gaming machine for weeks at a time to perform all-atom simulations of the 30S subunit.

Finally, I would like to thank my family: my wife Loryn, son Jack, mother Karin, father Bob, brother Robert and sister Mirabai, whose love have been the source and ultimate meaning for my life and this work.

Table of Contents

Part II: Methodology

List of Tables

List of Illustrations

Part I: Background

Chapter 1

Introduction

This dissertation presents theoretical research on the structure and function of the ribosome. The primary results have been the creation of a software program called 'The Ribosome Builder' , which demonstrates new methods for creating models and simulations of macromolecular systems such as the ribosomal translational machinery.

1.1. Two Code-Processing Machines

The research presented here has been involved with two separate things: one which is very ancient and one which is still quite new. Although a vast span of time separates them, they share a fundamental power. The two things are the ribosome and the digital computer. One assembles molecules and the other manipulates electrons but they share the ability to function as programmable machines that operate upon a code.

These machines in and of themselves are quite complex and extraordinary, but their true significance lies with the inputs and outputs of their activity. As code-processing machines, the emergence of both the ribosome and the computer resulted in the growth of large amounts of new information and products. In the case of the ribosome, the information took the form of genomes, and the products were the vast array of protein molecules and their activities. For computers, the information took the form of computer software, and the products are the many activities performed by modern digital computer

systems. The real significance of these code processing machines is the transition that each produced from an existing way of life to a new and more powerful one.

To understand the reasons why such powerful transitions occurred, we can look to the early history of computers for insight. When computers emerged with the ability to read a code and to produce something from it, an interesting thing happened. Because the cost of storing information is a relatively minor expense relative to other parts of the system, a lot of code tends to accumulate rather quickly, and seemingly with little effort. The acquisition of truly effective and efficient code is another matter, and much time and effort may be required for its development. But once the ball starts rolling in a technology that uses code, it tends to keep rolling. Code for new capabilities tends to accumulate and battle-tested refinements of existing code tend to be retained.

It is likely that the period of time following the appearance of the first functional ribosomes was one of extraordinary change and activity, in comparison to the age that preceded it. The precise details of this emergence are entirely unknown at present, but it seems certain that the new ability to assemble amino acids into proteins, under direction from a nucleic acid sequence, would have resulted in an explosion of new molecules and interactions. If, at the time, a system of molecules or protocells could have felt excitement and wonder, this would have been a time for such feelings. As human beings are most certainly capable of such feelings, we can reflect upon the birth of the ribosome in the primordial world and compare it to the current period of intensely exciting activity and growth that has followed the emergence of the digital computer in modern times.

1.2. A Brief Outline

This dissertation presents scientific theory and computer technology applied together toward the primary goal of understanding the structure and function of the ribosome. It is organized in three parts: background, methodology and applications. The background begins with specific aims of the project, along with larger goals and a brief history of the work done. The rest of the background gives some brief coverage of the theoretical and experimental methods that have been relevant to the actual research. The next part of the dissertation presents the research methodology. This is a description of the implementation details of a software program called 'The Ribosome Builder', which has been the primary product of the research. The final part presents four applications of the Ribosome Builder program. These consist of molecular modeling and simulation work on nucleic acids and the ribosomal translational machinery. Specifically, they are: 1) structural alignment of ribosomal subunits, 2) folding of an RNA tetraloop, 3) wrapping mRNA to a path through the ribosome, and 4) a schematic simulation of the elongation cycle of translation. The dissertation concludes with an assessment of the work done and considerations for future research.

Chapter 2

Goals

2.1. Understanding the Ribosome

Much has already been achieved towards understanding protein translation. The overall process has been clear for quite some time, in which interactions between messenger RNA and transfer RNAs take place on the framework of two ribosomal subunits, leading to the sequence-specific assembly of amino acids into protein. A great deal of biochemical, genetic and structural experimental data have been amassed, and detailed models of individual parts and sequences have been assembled.

Research on the ribosome is ongoing, and new results are continually being obtained to contribute to the existing body of knowledge. However, because of the extraordinary complexity of the subject, there is a growing need to unify the existing data and isolated models into a comprehensive, detailed and unified model of the entire process. Our current understanding of the overall process is represented through a diverse number structural models and detailed interactions. But unification of the collection of these models and data into a complete, rigorous and explicit form is still waiting to be done. The primary goal for the research for this dissertation has been to develop this type of unified model.

The central element of the dissertation is a hypothetical model of a complete cycle of elongation on the ribosome at an atomic level of detail. The model includes all of the

principal molecular components of translation, including the small and large ribosomal subunits, messenger RNA, transfer RNAs, and elongation factors EF-Tu and EF-G. The model proposes a detailed sequence of events that define movements of the smaller components to and from specific locations on the ribosome, as well as some conformational changes of these components during key steps in the elongation cycle. The model is provisional, uncertain and incomplete. However, these deficits are qualified by the urgent and growing need to break new ground in the development of comprehensive and dynamic atomic resolution models of complex macromolecular processes.

2.2. Creating Modeling Software

The research for the dissertation has occurred over an interval of approximately five years. During that time, the work was strongly influenced by a culmination of events in two distinct fields. Forty years of research on the ribosome by many investigators had accumulated to produce deep insights, including for the first time, atomic structural models of complete ribosomal subunits. During this same period, computer technology had advanced to the point where powerful desktop computers with 3D graphics were available at very low cost. As a result, computers capable of doing molecular modeling of the ribosome were becoming ubiquitous.

In the context of these events, a secondary goal of this work has been the development of computer software to enable individual researchers to analyze the newly emerging structural data and to integrate it into the existing body of knowledge about the ribosome. The software capabilities required for such analysis include 1) efficient storage and

representation of data, 2) graphical presentation of structures, 3) modeling and simulation of the chemical properties, and 4) effective user interfaces, with supporting tools and documentation.

A number of software tools already exist for constructing, analyzing, displaying and simulating molecular models. However, the newly emerging challenge of modeling the ribosome and other large biological macromolecules brings with it new requirements which have not been adequately addressed by much of the existing software. First, the large size of these structures requires considerable efficiency for interactive use and reasonable simulation times. Second, the integration and customization of software tools and data is becoming a more important requirement for the individual investigator. Finally, the way in which scientific software itself is optimally developed is crucially dependent upon certain philosophies that facilitate openness and collaboration (Cornell et al., 1995), (DeLano, 2005). There are many scientific software programs in common use that are proprietary, closed-source, and in some cases actually withheld from certain researchers (Giles, 2004). Such closed-source philosophies and practices are fundamentally at odds with the open nature of the scientific process (Benkler, 2004). Because of the importance of open-source principles to scientific research, a particular effort has been made in this dissertation to document the design and implementation of the Ribosome Builder software. Software programs are becoming much more complex, and this increases the need for better descriptions of the software implementation in order to allow other researchers to reproduce and extend the work that has been done. This need has been addressed in this dissertation through the liberal use of UML diagrams and other implementation details of the Ribosome Builder software, as presented in the

methodology section. In addition, the desire to make the results of this research more accessible to software developers has motivated a more extensive presentation of the theoretical and experimental background in the relevant biological fields of research. The requirements of efficiency, integration and openness are familiar and worthwhile reasons for development of better scientific software. However, there is another more specific and immediate goal that has determined the development of the Ribosome Builder software. This goal is concerned with solving something known as 'the timestep' problem, because it has to do with an important limitation in existing molecular dynamics software where the activity that can currently be simulated is confined to very small time intervals compared to what is needed (Schlick et al., 1997). A significant portion of the Ribosome Builder project is a methodology for overcoming this timestep limitation.

2.3. Larger Goals

2.3.1. Multi-Scale Modeling

In close connection with achieving the goal of longer simulation times in molecular dynamics, there is a larger goal in biological modeling. This goal is 'multi-scale' modeling (Hunter, 2004), which is concerned with creating biological models that span multiple size and time scales. There are well developed methods for representing objects at particular size scales. Quantum mechanics is applied at the level of electrons and nuclei. Molecular mechanics and molecular dynamics model behavior at the scale of atoms and molecules. At the more macroscopic cellular and multicellular levels, objects can be modeled with methods based on thermodynamics. Things modeled at these levels include metabolic pathways and gene expression patterns.

At the interface between the microscopic and the macroscopic level, things get a little interesting. The number of atoms involved currently preclude methods that rely on their explicit representation. But macroscopic methods that employ bulk concepts such as finite elements and mechanical stress may fail to capture essential microscopic interactions. The techniques for hybrid structural modeling that can bridge the microscopic and the macroscopic worlds have yet to be fully developed (Hunter and Borg, 2003). With the building avalanche of genomic data and the consequent discoveries in the proteome, cell biology, developmental biology, immunology and countless other areas of physiology, the need for multiscale modeling techniques will become imperative. The alternative may well be to drown in a rising sea of unexploited data that surrounds disconnected islands of scientific theory.

2.3.2. Enhanced Simulation and Informational Environments

Identifying important goals, such as the attempt to model everything from atoms to elbows, is all very well. The next question is how to go about achieving them. One area appears quite promising as a source of new technologies for multiscale biological modeling. This area lies in a yet-to-be-fully-defined intersection between networked 3D computer game technologies and persistent online virtual worlds. A speculative vision of this intersection is an immersive 3D virtual environment that serves as a workplace for construction, comprehension and communication of multiscale biological models. A brief argument in support of this vision is that multi-scale structural and dynamic models will be most effectively developed in a compatible technological context. This context includes components such as 3D graphical interfaces, physics engines, and multi-user

content-creation tools. The promise of 3D virtual world technologies for scientific research and collaboration has been one of the motivations for the work done in this dissertation.

2.3.3. Understanding the Organism

It almost goes without saying that creation of integrated models of living organisms at multiple levels will have important benefits and implications. Further advances in medicine are crucially dependent on such models. Economic benefits from biotechnology are likely. Most importantly perhaps, is the hope for understanding the physical basis of human cognition. This scientific pursuit joins hands with philosophical and spiritual quests towards a greater understanding of ourselves and the world.

Chapter 3

Project History

3.1. Original Proposal

The original dissertation proposal was titled 'Molecule Theatre, A Parallel Molecular Dynamics Program for Modeling RNA and DNA', submitted in September, 1999. The proposal was to develop a software program employing combinations of molecular dynamics and quantum mechanics to simulate nucleic acid activity. The implementation would employ parallel processing and a 3D graphical interface.

The proposed simulation approach, termed 'Orbital Mechanics' (OM), aimed to represent molecular activity in a way that was intermediate in scale between Quantum Mechanics (QM) and Molecular Mechanics (MM). Quantum Mechanics was a more rigorous and exact method of simulation, but with a considerable performance penalty. Molecular Mechanics required the use of a set of empirically derived parameters that were less exact and less generally applicable, but enabled simulation of much larger molecular systems. The OM approach was to explicitly simulate the electronic structure of atoms, but in a simpler way than QM. The main goal was to have an ab initio means of simulating nucleic acid folding and movement that would be more general than molecular mechanics, and not tied to a particular set of empirically derived parameters. The expectation was that this approach would be inherently less computationally intensive than QM, but still somewhat more intensive than MM. Because the ultimate targets for

the new method were very large nucleic acid structures such as the ribosome, a parallel processing implementation was intended to handle the huge model sizes.

The details of the OM approach involved the representation of localized electronic charges in valence shells around the nuclei of atoms. These charges were tethered to fixed radii, but allowed to reorient at different spherical angles around the nuclei under the influence of a coulomb force function. The interaction forces between the tethered electrons were transmitted and summed at the nuclei, and standard equations of motion were used to update the positions of the nuclei at each timestep.

3.2. Summary of Work Done

During the course of research, the molecular modeling work went through a continual process of evolution as experience was gained, mistakes were made, and lessons learned. During this interval, covering the time from Fall 1999 to Fall 2004, the work can be divided into three distinct project phases: (1) Molecule Theatre, (2) Ribosome Builder, *glut* implementation (OpenGL Utility Toolkit), and (3) Ribosome Builder, *Qt* implementation. Parts of the work history are summarized below. There are also project log files with more detail, especially after April, 2003. The project logs are available at http://rbuilder.sourceforge.net/devel.html#logs. A working version of the program is available at http://rbuilder.sourceforge.net on the download page. The version of the program that is referenced in this dissertation is version 0.3.4, released on January 11, 2005.

3.2.1. Molecule Theatre (1999 - 2000)

During the work on Molecule Theatre, a forcefield was developed and was able to simulate the activity of small molecules, up to the size of single nucleotides. The simplified representation of electronic structure was able to successfully demonstrate certain realistic features, such as the formation of tetrahedral bond geometry in methane. Stable bonding of larger structures, such as single nucleotides, was also observed. However, there were several problems. The first problem was that the structures did not maintain rigid bond angles. Playback of movies recorded from the simulations showed them to be very 'floppy', with hydrogen atoms especially so. The problem was that valence electrons were represented as point sources instead of distributed in space, and were artificially confined at the center of simplified orbitals. A better appreciation was gained for the extraordinary forces that act between atoms to produce bond lengths and bond angles that undergo extremely small deviations, relatively speaking. Huge forces must act at extremely short distances and extremely small intervals of time in order to maintain mostly rigid bond angles.

The other problem that emerged was the very small timestep interval needed to properly simulate fluctuations in electronic structure. The program employed a method of automatically adjusting the time scale until a stable simulation was obtained. The 'natural' time interval that emerged was on the order of 10^{-18} seconds. This extremely small timescale, and the great amount of computation that was required, even for small molecules, did not provide hopeful prospects for simulation of Ribosome-sized models unless a great deal of parallel processing power would be available.

3.2.2. Ribosome Builder, *glut* implementation (2000 - 2002)

In response to the scaling problems encountered in representing fine electronic structure in Molecule Theatre, an effort to directly support large models was made in the first implementation of Ribosome Builder. Explicit representation of atoms was maintained, but the covalent interactions were simplified by consolidating groups of atoms into rigid 'blocks' connected by spring bonds. The hope was that by skipping many of the covalent interactions, the program would be efficient enough to simulate entire ribosomal subunits. During the summer of 2000, atomic resolution models of the ribosome became available for the first time, so the decision to change directions to support larger molecular models turned out to be a good one. The code for loading PDB files was improved to support rapid loading of the very large models of the ribosomal subunits (50,000+ atoms). After support for these PDB models was added, the creation of a simple molecular mechanics forcefield was begun, with a strong emphasis on efficiency. Because steric interactions between individual atoms were still being explicitly modeled, a number of third party collision detection libraries were investigated, with the hope of lowering the number of pairwise interactions that had to be calculated. The SWIFT++ (Ehmann and Lin, 2001) and VCollide (Hudson et al., 1997) libraries were tried out, with only limited success in terms of performance. A number of custom hierarchical methods were also attempted, and again the performance gains were less than desired.

While work on the forcefield was continuing, several other areas were explored. One area concerned interactivity. The user was given the ability to interactively adjust the positions of models while the forcefield was running. To complement this capability, visual display

of the molecular forces was added. This display consisted of graphical vectors showing the direction and magnitude of the forces. A further experiment in interactivity and immersive simulation was attempted by adding an option to generate sounds during atom collisions. The frequencies of the sounds were a function of the magnitudes of the forces and were updated every timestep. The results were novel, but somewhat disturbing to the ear.

3.2.3. Ribosome Builder, *Qt* implementation (2002 - 2005)

One of the lessons learned from Ribosome Builder (*glut*) was that use of 'rigid blocks' to represent groups of atoms was too severe of an approximation. Too many invalid structural deformations were observed with the approach, and it became clear that it would be necessary to represent rotations about every covalent bond within the model. The challenge of doing this efficiently, while maintaining the very small tolerances in covalent bond distances and angles, was ultimately achieved through the use of the 'shadow point' method in Ribosome Builder (Qt).

Another area where significant changes were made was the user interface for Ribosome Builder (Qt). The first implementation had a graphical interface built on glut, the OpenGL utility toolkit (Robins, 2001). Unfortunately, the toolkit was too limited in its support for interface tools, menus and dialogs. The growing number of program features required better controls and feedback for the user. This was achieved by re-implementing the program using the Qt application framework (Trolltech, 2004). The Qt implementation of the program is still the one in use at this time, so additional discussion of its implementation history is deferred to the methodology section below.

Chapter 4

Quantum Mechanics

"we assert that the atom in reality is merely the diffraction phenomenon of an electron wave captured as it were by the nucleus"

(Schrodinger, 1933)

4.1. Introduction

Although quantum mechanics (QM) methods have not been used directly in the Ribosome Builder research, quantum theory is the ultimate mathematical foundation for many kinds of molecular modeling. The application of quantum mechanics to problems in chemistry is referred to as 'quantum chemistry' (QC). Related terms are 'Ab initio' and 'non-empirical' modeling. Quantum chemistry methods are used extensively in parameterization of molecular mechanics forcefields (Cornell et al., 1995), Quantum Mechanics/Molecular Mechanics hybrid modeling (Morokuma, 2002), interpretation of NMR experimental data (Oldfield, 2002), and chemical reaction dynamics (Schlegel, 2003). Recently, QC methods have also being applied directly to larger size biomolecular models for geometry optimization (Van Alsenoy et al., 1998). This is especially significant for nucleic acids, because it has typically been more difficult to accurately model nucleic acid structures with molecular mechanics methods (Hutter et al., 1996). For these reasons, it is appropriate to briefly review some of the basic principles of ab

initio quantum theory, its relation to less exact computational methods, and its current limitations and future prospects in molecular modeling.

Quantum Mechanics arose in the early 20th century from the work of Planck, Bohr, Heisenberg, Schrodinger and Dirac, among others. Shortly afterward, the tools of quantum theory were applied to chemical problems. A nice summary of the history, methods and applications in quantum chemistry is given by Barden (Barden and Schaefer, 2000). While many of the theoretical foundations for quantum chemistry were laid down in the 1930's shortly after the birth of quantum mechanics, the amount of computation required was excessive for much practical application. The emergence of powerful digital electronic computers in the 1950's and 1960's enabled the development of algorithms and software to perform quantum chemical computations. Because the computational burden remained (and still remains) very formidable, even with constant advances in computing power, there is very active and ongoing research to yield more accurate and efficient computations.

4.2. Schrodinger's Equation

The Schrodinger equation (SE) is one of the most fundamental mathematical expressions for describing the interactions between electrons and atomic nuclei. The non-relativistic time-independent form of the SE is shown in Equation 1, which is a differential equation involving the quantum mechanical Hamiltonian operator H applied to a wave function Psi. A wave function which satisfies this equation is called an eigenfunction because the right hand side of the equation is a constant multiple of the left hand side. The constant

value E is called the eigenvalue, which corresponds to the total energy associated with a solution of the wave function.

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

The complete wave function Psi is a stationary state function of the spatial and spin coordinates of N electrons and M nuclei, as shown in Equation 2. It is a complex function, but the square of its amplitude is a real value that represents the probability of the spatial locations of the electrons and nuclei in the system (Woodbury, 1997).

$$\Psi = \Psi(r_1, r_2, \ldots, r_N, A_1, A_2, \ldots, A_M) \tag{2}$$

Equation 3 is the electronic Hamiltonian, which can be used when the Born-Oppenheimer approximation is made (Szabo and Ostlund, 1989). This approximation keeps the positions of the nuclei constant when calculating the electronic wave function. The approximation is justified in many situations because of the assumption that change in electronic structure occurs much more rapidly than movements of the much heavier nuclei. The reduction in variables greatly simplifies the problem of finding solutions for the wave function. There are cases where the Born-Oppenheimer approximation breaks down, such as the competing bond fission reactions of bromoacetyl chloride (Butler, 1998), but for a large portion of quantum chemistry the assumption is valid.

$$H_{elec} = \sum_{i=1}^{N} \frac{1}{2} \nabla_i^2 - \sum_{i=1}^{N} \sum_{A=1}^{M} \frac{Z_A}{r_{iA}} + \sum_{i=1}^{N} \sum_{j>i}^{N} \frac{1}{r_{ij}} \tag{3}$$

4.3. The Challenge to Solve the Equation

The Schrodinger Equation cannot be been solved exactly for anything beyond simple one-electron systems, and the challenge in the 20th century has been to develop methods that give approximate solutions. There has been some success in meeting this challenge, as indicated by the recent Nobel prize in Chemistry awarded to Walter Kohn and John Pople in 1998. The Wave Function methods of Pople (Pople, 1998) and the Density Functional Theory of Kohn (Kohn, 1999) form the two distinct branches of quantum chemistry in use today, and an overview of these methods will be briefly discussed.

4.3.1. Wave Function Methods

The difficulty in solving the Schrodinger equation for multi-electron systems lies with the electron-electron interactions in the potential portion of the electronic Hamiltonian. Hartree-Fock theory was an early approximation that involved replacing multi-electron interactions with a single 'effective' potential that is experienced by each individual electron. This made the equation separable, and thus possible to solve numerically, but for larger systems the difficulty of integration was still intractable (Van Houten, 2002). An important advance was made in the 1950s with the use of gaussian functions to represent the basis sets, instead of 'Slater-type' orbital functions (Pople, 1998).

Basis sets are groups of mathematical functions that are used to create molecular orbital wave functions. Modern basis sets typically contain groups for both core (inner shell) electrons and valence electrons. The name of a basis set can reflect the number and type of primitive functions used to construct the orbitals. For example, the 6-31G* basis set, which is commonly used for molecular mechanics development, is one in which the core

orbitals are constructed from 6 primitive gaussian functions and the valence shell orbitals are constructed from two parts (split-valence). The first part is defined by 3 primitive gaussian functions and the second part is defined by 1 primitive gaussian function (Binkley et al., 1980). The asterisk indicates that polarization functions have been added (Pulay et al., 1979).

Developments in the construction of basis sets enabled the practical solution of wave functions for larger molecules, but the Hartree-Fock approximation still results in significant error because it fails to account for the effects of electron-electron interactions. Several categories of theory were created to address these interactions: Configuration Interaction (CI) theory, Coupled-Pair theory and Many Body Perturbation (MBPT) theory (Szabo and Ostlund, 1989). From each of these main categories arose a family of acronyms that represent the many particular methods that are commonly used, such as FCI, CID, CISD for CI theory; CCD, CCSD and CCSD(T) for Coupled-Pair theory; and MP2, MP3 and MP4 for Perturbation theory. Many of the differences in these methods reflect trade-offs between accuracy and computational cost.

4.3.2. Density Functional Theory (DFT)

Density Functional Theory (DFT) is an increasingly popular method for calculating ground-state electronic structures and energies from first principles (Argaman and Makov, 2000). DFT uses an approach that is quite distinct from the wave function methods that are based on Hartree-Fock theory. Instead of solving for a wave function of multiple electrons, DFT formulates the problem in terms of the electron density, which greatly reduces the number of coordinates, and consequently, results in significant

computational advantages over many wave function methods of comparable accuracy (Kohn, 1999).

The foundations for DFT were laid in the early 1960s, first, with the paper by Hohenberg and Kohn, which proved that the electron density function n(r) uniquely determines the non-degenerate electronic ground state (Hohenberg and Kohn, 1964). Then in 1965, Kohn and Sham developed a set of equations that can be used as a practical procedure for calculating the electron density and related energies for a system of electrons (Kohn and Sham, 1965). These equations, referred to as the Kohn-Sham equations, are somewhat analogous to the self-consistent equations in Hartree-Fock theory in that they are solved iteratively to produce better approximations to the density distribution (Capelle, 2003). Because of the greater computational efficiency, DFT methods have been applied to larger sized molecules including many biological structures (Andreoni et al., 2001). These include nucleic acid base pairs and surrounding waters and ions (Hutter et al., 1996), (Guerra et al., 2000), liquid water (Sprik et al., 1996), and organometallic structures (Mingos et al., 1995). In addition to pushing the size envelope, these DFT studies serve as important points of reference for evaluating the accuracy of other theoretical methods, at both the ab initio and empirical level.

4.4. Hybrid Methods

DFT modeling has also been combined with molecular dynamics methods to yield a hybrid approach called 'Ab Initio Molecular Dynamics' (AIMD) (Tse, 2002), also called Car-Parinello Molecular Dynamics (CPMD), after the introductory paper on the method (Car and Parrinello, 1985). The hybridization of ab initio and empirical methods has been

an active area of research and aims to greatly extend ab initio theory to ever-larger systems. The difficulty of properly integrating the two types of computations can be quite formidable however. One of the principal difficulties is how to define the interface between the QM and MM portions of a simulation. As a result, attempts to systematize and evaluate hybrid QM/MM modeling have been developed, such as the ONIOM framework (Morokuma, 2002).

4.5. Software

Software for quantum chemistry has come a long way since the medieval ages of the 1950s, where vacuum tube monsters had to be carefully watched all night in case they started belching smoke! (Pritchard, 2001). There are a number of open source software programs currently available for ab initio calculations. Programs for wave function methods include GAMESS (http://www.msg.ameslab.gov/GAMESS/GAMESS.html), MPQC (http://mpqc.org), and Dalton (http://www.kjemi.uio.no/software/dalton/dalton.html). Software codes that focus on Density Functional Theory include ABINIT (http://www.abinit.org), octopus (http://www.tddft.org/programs/octopus/) and PWSCF (http://www.pwscf.org/).

The mathematics involved in quantum mechanical calculations of electronic structure can be truly formidable and daunting for the general population of scientific researchers. However, use of the theory holds out the possibility for acquiring truly profound insight into the actual chemical behavior of molecules. Computer software is increasingly able to exploit and encapsulate the complex, powerful methods of quantum chemistry and deliver practical results to the user.

Chapter 5

Molecular Mechanics

"The objective of molecular mechanics is to develop a simple model with which to represent the behavior of molecules."

(Allinger and Durkin, 2000)

5.1. Introduction

Molecular Mechanics (MM) is a general category of computation for determining chemical energies and structures. It is simpler and less exact than Quantum Chemistry. Instead of solving for electron wave functions or densities, the potential energy of a molecule is described with a set of simple functions that represent bond lengths, bond angles, steric forces and point charges. The formulation is also referred to as an 'Empirical Forcefield'. Because of their simplicity relative to ab initio methods, Molecular Mechanics forcefields enable the simulation of much larger systems of atoms, on the order of 100,000 atoms or more (MacKerell, 2004a). Empirical forcefields are used to predict static conformations of molecules via energy minimization procedures. They are also the basis for calculating conformational changes in Molecular Dynamics simulations. The fundamental components of MM forcefields will be covered here, in conjunction with the difficult problem of parametrization. This will be followed by a brief discussion of the related issues of long range effects and solvation. The background on MM will

conclude with a brief survey of some current biomolecular forcefields, their existing limitations and future prospects.

5.2. The Potential Energy Function

The central feature of empirical forcefields is the potential energy function. A typical function is shown in Equation 4, taken from the 1995 version of the AMBER forcefield (Cornell et al., 1995). This composite energy function is composed of terms which capture different aspects of molecular behavior. The terms include bond length stretch, bond angle bending, dihedral barriers, van der Waals interactions and electrostatic charge.

$$E_{\text{total}} = \sum_{\text{bonds}} K_r(r - r_{\text{eq}})^2 + \sum_{\text{angles}} K_\theta(\theta - \theta_{\text{eq}})^2 + \sum_{\text{dihedrals}} \frac{V_n}{2}[1 + \cos(n\phi - \gamma)] + \sum_{i<j} \left[\frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} + \frac{q_i q_j}{\epsilon R_{ij}}\right] \quad (4)$$

Atom coordinates, in conjunction with parameters that are specific to particular atom types, are used to evaluate the individual energy terms, for all pairs of interacting atoms. The resultant total energy function can be used directly in energy minimization calculations. Energy minimization is an attempt to find the most stable conformations of a molecule, which consequently will be the most probable ones that a molecule will adopt. The most stable conformations are simply those with the minimum total energy. A second important application of the potential energy function is to obtain its gradient, which determines the forces acting on the atoms. These forces are then used to calculate the movements of atoms over time in molecular dynamics simulations.

5.2.1. Bond Lengths

The bond length component of the potential energy function is a simple harmonic potential that is defined by a spring force constant K_r and equilibrium bond length r_{eq}. This equilibrium bond length is only an ideal. Actual equilibrium bond lengths are exquisitely sensitive to the local environment. Even slight rotations about a dihedral angle will correlate with small changes in the equilibrium bond length (Allinger et al., 1996b). Thus, the bond length term works well as a first approximation for maintaining the proper distance between atoms, but it is often supplemented with additional functions and cross-terms for greater accuracy.

5.2.2. Bond Angles

The bond angles of a molecule, in conjunction with the bond lengths, are the primary determinants of molecular structure. Although the instantaneous bond lengths and angles are altered by continuous vibration, the deviations from equilibrium values are typically quite small, on the order of fractions of angstroms and degrees, respectively (Allinger et al., 1996a). Consequently, this stability is responsible for the persistent structures and shapes of molecules. The specific values that bond angles adopt are fundamentally determined by the geometrical features of the molecular orbitals that result upon covalent bonding.

The molecular mechanics function that maintains the bond angle is usually a harmonic function similar in form to the bond length function. The potential is defined by the angle between three atoms, where two atoms are bonded to a common central atom. $Theta_{eq}$ is

the equilibrium angle and K_{theta} is the force constant. The relationship between the two terminal atoms of the bond angle is sometimes referred to as a 1,3 atom interaction.

5.2.3. Dihedrals

A dihedral angle involves four atoms A, B, C and D, bonded in a linear sequence. The angle itself is actually defined by the angle between the normals of the two planes (A, B, C) and (B, C, D). The dihedral angle reflects the conformational change that occurs with rotation about the B-C covalent bond. Bond lengths and bond angles typically undergo only fairly small fluctuations about their equilibrium values. Most of the significant conformational change in molecules, especially biomolecules, occurs through rotations about covalent bonds, which are correlated with changes in the corresponding dihedral angles.

The potential of the dihedral angle derives in part from the steric interactions between the terminal atoms A and D. However, other effects may also be involved, such as valence forces and hyperconjugation. The ultimate origins of the barriers to rotation remain controversial after many years of research, and as a result the dihedral potential has been characterized as the 'Bermuda Triangle' of electronic structural theory (Goodman et al., 1999).

In the dihedral summation term of Equation 4, V_n is the dihedral force constant and phi is the actual dihedral angle. Gamma is a phase angle of either 0 or 180 degrees (Duan et al., 2003). The function is a simple Fourier series that attempts to reproduce the periodic barriers encountered during rotation.

5.2.4. van der Waals Interaction

The van der Waals interaction is represented by a Lennard-Jones function, which consists of the first two terms in the final summation of Equation 4. This non-bonded interaction consists of an attractive force with an R^{-6} exponential and a repulsive force with an R^{-12} exponential, for two interacting atoms i and j. The large exponential values reflect how the magnitudes of these forces drop off very rapidly at larger distances.

The repulsive force dominates as the atoms closely approach each other. The repulsion is caused by the intersecting clouds of the inner shell electrons, which cannot occupy the same space and spin coordinates, in accordance with the Pauli exclusion principle. The attractive term dominates for distances greater than the sum of the van der Waals radii, and although the R^{-6} value is much smaller than the R^{-1} value of the electrostatic Coulomb term, the attractive van der Waals forces can be very significant in biomolecules. This is because very close packing of large numbers of atoms can occur at complementary surfaces. The source of the attractive forces, also called London or dispersion forces, results from induced dipoles that form during rapid fluctuations in the electron density (Schlick, 2002).

5.2.5. Electrostatic Charge

The final term of the final summation of Equation 4 represents the electrostatic interactions between atoms. The electrostatic potential is calculated using the standard coulomb function and point charges located at the atom centers. In order for this approach to be effective, the distribution of the discrete atomic charges must approximate to some degree the actual continuous distribution of the electron density in the molecule

(Momany, 1978). The determination of these partial atomic charges is one of the parametrization problems discussed below.

5.2.6. Supplemental Functions

In the energy function above (Eqn 4), there is no specific term for hydrogen bonding. Instead, the van der Waals and electrostatic parameters are adjusted for hydrogen bonding atoms to produce the equivalent behavior. Other forcefield versions may include explicit terms for hydrogen bonding. Supplemental terms can also be added to the function to give greater accuracy. These include cubic and quartic terms to extend one of the basic functions, such as bond stretch (Halgren, 1996c). There are also cross terms, which modulate the basic functions. One example is the 'stretch-bend' term, which correlates bond stretch and angle bending (Nevins and Allinger, 1996c). Finally, there additional kinds of fundamental terms that can be added. These include Urey-Bradley terms and 'impropers'. Urey-Bradley terms specify the distance between 1,3 atoms and are used to improve the geometry and vibrational frequency of bond angles (MacKerell et al., 1995). Impropers, also called out-of-plane bending, involve interactions between the three orbitals of the same carbon atom (Ermer and Lifson, 1973).

5.3. Parametrization

5.3.1. Goals

Forcefield parametrization is the task of defining the specific values for the equilibrium bond lengths, angles, force constants, and other parameters in the potential energy

function. The parametrization process is driven by a number of interrelated goals. One important goal is to produce accurate energies. A second goal is concerned with transferability, which relates to the range of molecules, and their conformations, that will be properly represented by the forcefield.

Accuracy is a problem in parametrization because, as stated previously, the potential energy function of molecular mechanics is a composition of functions, with each function trying to capture, in some simple way, a distinct aspect of molecular behavior. In general, however, the actual molecular behavior cannot be neatly partitioned in this way. Consequently, the calculated energy will always be an approximation to the true energy. The desired transferability of a forcefield will also affect the choice of parameters. The earliest forcefields were only capable of handling hydrocarbon molecules, because the lack of polarization greatly simplified the problem (Allinger et al., 1967). Later, with more sophisticated energy functions and parameters, forcefields were developed for additional types of molecules such as proteins, nucleic acids and carbohydrates. Because of the large number of careful judgements and decisions that must be made in the process, the development of a forcefield has been aptly characterized to be as much an art as a science (Halgren, 1996a). In frank recognition of its somewhat esoteric nature, (Cornell et al., 1995) emphasized the need for more explicit descriptions of the parametrization process. The hope was to make it easier for future development and extensions, by the original authors as well as by independent researchers. This is an important goal, and has been successful to some degree. A notable advance is observed in (Wang et al., 2004), but there is still room for improvement.

5.3.2. Procedures

The process of specifying of parameter values typically begins with drawing upon a variety of experimental data and ab initio calculations. A small set of model compounds must be chosen, and the parameters obtained from this small set should be applicable to a more general set of molecules. The starting parameters are generally adjusted through an iterative process to give greater consistency with the source data. Finally, the resulting forcefield may be put through a series of tests on representative compounds in order to evaluate its accuracy, and to compare the results with that of other forcefields.

The initial values for equilibrium bond lengths and angles are commonly taken from crystal structures. The stretch and bend force constants are obtained via normal mode analysis of vibrations in spectroscopic data (Cornell et al., 1995). van der Waals parameters are developed from thermodynamic data such as liquid densities and heats of vaporization (Jorgensen et al., 1984). These experimental values are supplemented by ab initio calculations. The model compounds used include propane, butane and isobutane for representation of aliphatic hydrocarbons. Model compounds for proteins include N-methylacetamide and alanine dipeptides (MacKerell et al., 1998).

Determination of partial atomic charges is one of the more important yet problematic areas of parametrization (Halgren, 1996b). The charges are necessary for calculation of the electrostatic potential by the coulomb function. Quantum mechanical calculations enable the definition of a continuous electrostatic potential, from which a set of discrete atomic charges can be created that approximate it. Early approaches achieved this with a simple least-squares fitting procedure (Momany, 1978). Although a set of charges could

be found to reproduce the given potential, the results still had certain limitations. One issue is that of transferability, where the charges assigned to atoms may not work well when applied to the same atom types in a different molecule. Another issue is conformer dependence, where the set of fitted charges does not produce the desired potential when the molecule changes to a different conformation. A more complex fitting procedure, Restrained Electrostatic Potential (RESP), was developed by (Bayly et al., 1993) to address some of these issues. The method uses restraint functions and symmetry considerations during the charge fitting procedure to minimize excessive internal charges and provide greater conformer independence. The success of the approach by Bayly et al. motivated its adoption in the AMBER forcefield of (Cornell et al., 1995).

5.4. Long Range Interactions

5.4.1. Cutoffs

The non-bonded steric and electrostatic interactions, in contrast to the bonded interactions, can occur in principle between every possible pair of atoms in the system. For larges numbers of atoms, the $O(N^2)$ number of operations to calculate these pairwise interactions can become a computational bottleneck. For steric interactions, this problem is satisfactorily dealt with by applying some kind of 'cutoff' method, which skips interactions between atoms that are far apart. The types of methods range in complexity from simple truncation to more sophisticated approaches such as switch and shift functions. Truncation is an 'all-or-nothing' rule that allows interactions between atoms within a certain distance and then abruptly skips interactions beyond that distance. The abruptness of this truncation can give rise to certain artifacts because of the discontinuity

in the derivative of the potential energy function. Switch functions apply a reducing factor within a certain distance interval that reduces the potential to zero at the end of the interval. Shift functions modify the potential function throughout the entire range, from zero distance to the cutoff distance, so that it smoothly approaches zero at the cutoff distance (Schlick, 2002).

5.4.2. Long Range Electrostatics

The electrostatic interaction, with a 1/R dependence, falls off much more gradually than the steric interaction. Consequently, the long range effects are much more significant and applying cutoff methods can produce serious artifacts. This is especially true for nucleic acids, which have a net negative charge on the phosphate backbone. Unrestrained DNA and RNA helices can rapidly fall apart when applying standard cutoff methods in molecular dynamics simulations (Cheatham et al., 1995), although (MacKerell, 1997) provides a counter-example. Even the stability of small peptides, which have net dipole moments, may be adversely affected with simple cutoff procedures (Schreiber and Steinhauser, 1992). A huge amount of research has been done over the years to produce more sophisticated models of long range electrostatic interactions (Cramer and Truhlar, 1999), (Tomasi, 2004). The problem is further complicated by the interrelated nature of electrostatics, solvent models and polarization.

5.4.3. Polarization

Electronic polarization is the displacement of the electron clouds around an atom in response to an external electric field (Gilson, 1995). Second generation forcefields implicitly account for some solvent polarization effects through the use of the HF/6-31G*

basis set for calculating partial atomic charges. The basis set overestimates the dipole moments of peptides in the gas phase, with the result that the increased polarity of the point charges mimics the effect of polarization that would occur in the condensed phase (Duan et al., 2003). While this approach has been successful for predicting solvent polarization effects such as solvation energies and heats of vaporization, it does not account for polarization in the interior of proteins. A major goal of current forcefield development is to try to incorporate more sophisticated models of polarization (MacKerell, 2004a).

5.4.4. Solvent Models

Any treatment of electrostatics and polarization in condensed phase must take account of the solvent, which for biomolecular forcefields is usually water. There are two separate categories of solvent models: explicit and implicit solvent. Explicit solvent models define a set of parameters to represent water molecules and calculate direct interactions between individual water molecules and solute molecules. Some commonly used explicit solvent models include TIP3P, TIP4P, SPC and F3C (MacKerell, 2004a). Particular forcefields also tend to work best with particular solvent models because they were the ones used during the parametrization process of the forcefield.

Implicit solvent models attempt to represent the collective effects of solvent. These types of models can be further subdivided into continuum and discrete models. In addition to the computational advantages of having fewer explicit atoms in the system, implicit models can draw upon a more powerful theoretical framework for representing polarization (Cramer and Truhlar, 1999).

5.4.5. Computational Methods

An accurate and computationally efficient method for calculating long range electrostatics is the Particle Mesh Ewald (PME) algorithm. This algorithm built upon early work by Ewald that defined an infinite periodic array of cells and calculated the unit cell energy from two summations, one in real space and the other in reciprocal space. After corrections were added to handle nonuniform fields, an accurate method was obtained, but $O(N^2)$ calculations were required. Subsequently, fast fourier transform methods were used to approximate the reciprocal summation in O(N log(N)) steps (Darden et al., 1999). This resulted in a practical and widely used method for electrostatic calculations of macromolecules. The accuracy of the method was demonstrated by nanosecond-scale MD simulations that showed convergence of DNA helices to a common structure after starting from different conformations (Cheatham and Kollman, 1996).

Another important class of methods for electrostatic potential calculations are those based upon the Poisson-Boltzmann (PB) equation. The PB equation represents the electrostatic potential in terms of the charge density of molecules and the thermodynamic distribution of ionic charges in solution around those molecules (Fogolari et al., 2002). For simple geometrical shapes, such as spheres, cylinders and planes, analytic solutions of PB can be obtained. These shape approximations can be used in biomolecular applications for globular proteins, DNA helices and membranes, respectively. For more complex shapes, finite difference methods can be used to obtain numerical solutions of the PB equation. Common applications include calculation of the electrostatic potential at the solvent

accessible surface of biomolecules. One impressive example is the calculation of the electrostatic potential of the very large ribosomal 30S and 50S subunits, made possible by implementation of efficient parallel processing algorithms (Baker et al., 2001).

5.5. Survey of Current Forcefields

This background on Molecular Mechanics will conclude with a brief survey of current forcefields, some of their capabilities, limitations and future directions. Forcefields can be categorized by the types of molecules they can represent. There are forcefields primarily developed for hydrocarbons such as the older MM2 (Allinger, 1977), forcefields for biomolecules such as AMBER (Cornell et al., 1995), CHARMM (MacKerell et al., 1998), GROMOS (Scott et al., 1999), and OPLS (Jorgensen et al., 1996). For biomolecules, the primary development has been for proteins (MacKerell, 2004a), but there are also specialized versions of some of the major forcefields for certain types of biomolecules, such as the nucleic acid versions of AMBER (Foloppe and MacKerell, 2000) and CHARMM (MacKerell and Banavali, 2000). There are also forcefields developed with certain kinds of applications in mind, such MMFF (Halgren, 1996a), which is used in the pharmaceutical industry and places special emphasis on receptor-ligand interactions.

Modern molecular mechanics forcefields have made good progress towards accuracy. A recent evaluation of eight major forcefields (Halgren, 1999a) compared the predicted conformational energies of more than 30 different compounds to experimental values. In many cases, the predicted values were successfully estimated to within fractions of a kcal/mol, and few errors were greater than 3 kcal/mol. Of course, the choice of which

compounds to evaluate may play a role in how well a forcefield does. The forcefield developed by the authors of the paper just happened to perform the best out of all eight that were considered!

In addition to the ever present needs for greater accuracy and transferability, most modern molecular mechanics forcefields have difficulties representing metals and organometallic compounds (Gajewski et al., 1998). Another significant limitation occurs with chemical reactions, as covalent bond breaking and formation are generally not supported. Various methods for better approximations of polarization are currently being worked on, and this is likely to remain an active area of development for the next generation of forcefields (Ma et al., 2000).

Chapter 6

Molecular Dynamics

"The ability to calculate structure and free energy ... heralds the beginning of a fourth era of macromolecular MD."

(Kollman et al., 2000)

6.1. Introduction

Molecular Dynamics (MD) is a powerful theoretical technique for approximating the motions of atoms over time (Melchionna, 2004). MD methods are widely used to investigate biomolecules and are currently efficient enough to simulate system sizes of 10^4 to 10^6 atoms and time intervals of 10 to 100 ns (Karplus and McCammon, 2002), (Cheatham, 2004). The wide use of this technique is evident from applications in protein folding (Fersht and Daggett, 2002), enzyme activity (Daniel et al., 2003), x-ray crystal refinement (Brunger et al., 1998), nucleic acid structure and flexibility (Cheatham and Kollman, 1997), metal ion binding (Burkhardt and Zacharias, 2001), (Cates et al., 2002), and ribosomal movements (Sanbonmatsu and Joseph, 2003). Improvements in molecular dynamics accuracy and efficiency have been improving steadily in the last decade. Attempts to increase timescales and sampling are also being made with MD enhancements such as Targeted Molecular Dynamics (TMD) and Steerable Molecular Dynamics (SMD).

6.2. Fundamentals

The potential energy function of a molecular mechanics forcefield is typically used to determine the forces on a system of atoms. As shown in Equation 5, the force is defined as the negative of the gradient of the total potential energy of the system.

$$F(r_1, r_2, \ldots, r_N) = -\nabla V(r_1, r_2, \ldots, r_N) \tag{5}$$

With a force function, movements of atoms can then be calculated using Newton's equations of motion. The familiar second law, as shown in Equation 6, is a differential equation relating the acceleration of an atom to the total force on that atom. This equation is typically solved by numerical integration in the form of a finite difference equation. This is a discrete approximation which breaks up the smooth, continuous motion of atoms into abrupt movements that occur in a sequence of small timesteps. The Taylor series shown in Equation 7 provides a way to approximate the finite difference equation to various degrees of accuracy. One of the simplest approximations is the second-order Euler integration shown in Equation 8. Direct use of additional terms of the Taylor series for greater accuracy can be tedious because of the need to evaluate the higher-order derivatives. A popular alternative in physics-based simulations is the Runge-Kutta method, which works by approximating the higher derivatives (Bourg, 2002).

$$F_i = m\frac{d^2 r_i}{dt^2} \tag{6}$$

$$r(t+\Delta t) = r(t) + \frac{dr(t)}{dt}\Delta t + \frac{1}{2}\frac{d^2 r(t)}{dt^2}\Delta t^2 + \frac{1}{3!}\frac{d^3 r(t)}{dt^3}\Delta t^3 + \cdots \quad (7)$$

$$r_i(t+\Delta t) = r_i(t) + v_i(t)\Delta t + \frac{1}{2}\frac{F_i(t)}{m_i}\Delta t^2 \quad (8)$$

In many molecular dynamics codes, variants of the Verlet integration algorithm are used. This type of algorithm uses information from a sequence of adjacent timesteps in approximating the positions of the next timestep. Because the integration function in the Verlet scheme defines the next timestep in terms of itself as well as previous timesteps, this type of integration is termed 'implicit'. This is in contrast to the Euler integration, where the next timestep is only defined in terms of the previous timestep, and as a result, the function is explicit (Schlick et al., 1997). The Verlet algorithm has a straightforward implementation and produces well-behaved trajectories that remain close to the true Hamiltonian energy of the system (Schlick, 2002). Because of this well-behaved response and energy stability, Verlet methods are more commonly used than Euler or Runge-Kutta methods for molecular dynamics integrations.

6.3. Applications

There are numerous applications of MD in biomolecular simulation, but to best illustrate the use of MD that is relevant to this dissertation, the work in (Sanbonmatsu and Joseph, 2003) will be discussed. This work consists of molecular dynamic simulations of codon-anticodon interactions on the ribosome. This work attempted to build upon the

experimental work of (Ogle et al., 2002) in investigating near-cognate interactions. The simulation was motivated by the inability to observe native near-cognate interactions in the crystal structure, because they were unresolved.

The model consisted of approximately 60 residues, including the A-site codon, A-site anticodon stem loop, P-site codon and parts of the 16S rRNA near the codon-anticodon residues. The ends of the chain fragments in the simulation were tethered using harmonic spring forces. The residues were surrounded by a solvent consisting of 7000 water molecules. Simulations totaling 60 ns of simulation time were done to investigate interactions with a variety of single-mismatch codons, as well as a cognate codon interaction that served as the control. The results of the cognate model gave good agreement with crystal experimental data in the number of hydrogen bonds that were formed. Numerous differences were observed between the near-cognate and cognate simulations. These included changes in hydrogen bonding, changes in the depth of the minor groove, and increased solvent exposure of the near-cognate models relative to the cognate model.

These results could be analyzed quantitatively in terms of hydrogen bond count disruptions that occurred in the codon-anticodon-ribosome network. Quantitative changes in solvent accessible surface area were also calculated. The results tended to give support to the 'distortion detection' hypothesis for decoding (Lim and Curran, 2001). The work gives a brief indication of some of the useful results that can be obtained from molecular dynamics investigations of complex macromolecular systems. The limited activity that was investigated should be noted however. The conformational changes and number of residues involved were fairly small, and more than 60,000 hours of CPU time were

required. The work is a good start on investigating ribosomal dynamics, but a great deal more capability is needed for simulating global translational activity.

6.4. The Timestep Problem

When doing numerical integration with finite discrete timesteps, the magnitude of the timestep must be small relative to the most rapidly moving component in the simulation (Schlick et al., 1997). To gain better insight into the issue, one can consider the movements that must be simulated as a result of the interaction between two atoms. For example, in the hydrogen molecule there are two covalently bonded hydrogen atoms. For a condition in which the atoms are slightly displaced beyond the equilibrium bond distance, the MD integration must determine the positions of the atoms in the next timestep. If a very small timestep is used relative to the restoring force of the bond, then in the next timestep the atoms will be moved incrementally closer to each other as they try to return to the equilibrium distance. However, if too large of a timestep is used, the integration may produce a result where the atoms 'overshoot' the equilibrium distance and are positioned too close together, with the result that huge repulsive forces are created from the van der Waals interaction. This may cause the simulation to become unstable, and the atoms may fly apart or the calculations may fail completely due to numerical overflow in the floating-point representation.

(Schlick et al., 1997) gives an extensive review of numerous attempts to extend timestep length. A variety of approaches have been developed, with various degrees of mathematical sophistication. These include constrained dynamics, which remove the high-frequency movements through application of restraining forces. A common

implementation is the well-known SHAKE algorithm, and variants such as QSHAKE (Forester and Smith, 1998). Torsion Dynamics are a second class of extension methods. These involve the use of generalized coordinates which reduce the number of degrees of freedom so that only torsion angles are changing. Additional schemes include the Multiple Timestep (MTS) and Normal-Mode methods. Implementations of varying degrees of efficiency have been created for all of these schemes. However, the best increases in timestep length that have been achieved are fairly modest, not exceeding more than 30 or 40 fs. This is only within two orders of magnitude of the 1 fs lengths that are achieved with conventional integration methods, and thus falls far short of the improvements needed to investigate long-timescale activities such as protein folding.

6.5. Parallel and Ensemble Molecular Dynamics

There is another common approach for increasing the simulation times of classical molecular dynamics. This is more of a 'brute-force' approach that tries to distribute the workload among multiple computer processors. The general approach can be further subdivided into two categories: Parallel Molecular Dynamics (PMD) and Ensemble Molecular Dynamics (EMD). PMD involves the use of traditional high-speed supercomputers and parallel programming methodologies. Effective partitioning of the processing workload of a computation is not inherently trivial or straightforward. The ease with which a computation is parallelized often depends on the problem itself. A nice analogy is provided by (Pande et al., 2003): consider a graduate student thesis, which may typically be accomplished in 1500 days. Next consider the tantalizing prospect of

assigning 1500 graduate students to the same task. Can we expect the thesis to be completed at the end of one day? The answer is left as an exercise for the reader.

A partial success of the PMD approach is demonstrated by the work of (Duan and Kollman, 1998) in the 1 us simulation of the villin headpiece in solution. The simulation required 2 months of simulation time on a 256-CPU Cray T3D supercomputer. The success of the approach is qualified by the consideration that only a single trajectory was produced for a very small 36 residue peptide which inherently possesses one of the fastest known folding times (10 - 100 us). In addition, the 1 us simulation time required the use of relatively expensive and non-standard supercomputer resources. The performance of this approach is expected to improve substantially with the next generation supercomputers such as IBM's Blue Gene project (Allen et al., 2001) but the expense involved will still be considerable.

The strategy of (Pande et al., 2003) may be termed Ensemble Molecular Dynamics (EMD) and is distinct from classical PMD, which relies on efficient parallelization of a single molecular dynamics simulation. The guiding insight of EMD is the proposition that free energy barriers cause a conventional simulation to waste much of its time stuck in local energy minima. Only a small percentage of the time is actually devoted to simulating the transition of energy barriers to new conformations, which happen as a result of stochastic thermal fluctuations. The EMD strategy is to run multiple independent simulations, all starting from the same conformation, but with different velocities. If one of the simulations crosses an energy barrier to a new conformation, then all simulations are restarted from that single transition point, and the cycle is repeated.

The method has been successfully applied to simulate the folding of the C-terminal Beta-hairpin of protein G (Zagrovic et al., 2001). This involved 38 us of total simulation time that produced eight independent folding trajectories, using the distributed computing project Folding@Home (Folding website, 2005). This project makes use of thousands of volunteer personal computers connected through the internet. A more recent simulation involving the folding of an RNA tetraloop was accomplished with the same infrastructure, but with the equivalent of more than 150,000 CPUs (Sorin et al., 2005). The Ensemble Dynamics strategy makes it practical to use this kind of computing resource because of the independence of the multiple simulations. PMD simulations require much higher interprocessor communication speeds to be effective and would not be practical over the existing low-bandwidth internet.

Although standardization, cost and ease of access to supercomputing and distributed resources is expected to improve over time, the value of personal convenience and control of local computing resources should not be underestimated. This is especially true for experimental problem domains such as scientific research. The revolution in software development and use that attended the dawn of the personal computer era in the 1980s and 1990s is testament to the powerful advantages of local computing resources. Low cost, large numbers, convenience, flexibility and rapid development cycles all contributed to the success of personal computers at the expense of centralized mainframe solutions. Even if personal and flexible access to distributed computing resources improves substantially in the future with the use of server farms, high speed networks and virtualization of hardware and software environments, the inherent inertia of such distributed environments may continue to impose significant penalties on flexibility and

adaptability relative to local computing resources. Consequently, a 'desktop' capability for modeling and simulation should continue to remain an important consideration for scientific research in the foreseeable future.

6.6. Coercive Strategies

As efforts to lengthen the timestep have met with limited success and the potential for parallel computation has yet to be fully realized, other more unconventional strategies are being pursued for investigating long time scale activity. A number of these strategies may be loosely organized under the category of 'coercive methods'. The general idea is to apply some kind of external force or guiding information to the system in conjunction with the internal simulation of interactions. The expectation is that the external input can influence the system to sample distant conformations in much shorter amounts of time compared to uninfluenced simulations. However, a corresponding concern with this approach is whether the resulting simulations will be unnaturally 'biased' by the external input. In spite of such concerns, the results obtained can be useful. A variety of names are used to describe the specific techniques, and with ongoing development in this area, new terms will undoubtedly appear. The ones discussed here include Targeted Molecular Dynamics (TMD), Steerable Molecular Dynamics (SMD) and Interactive Molecular Dynamics (IMD).

6.6.1. Targeted Molecular Dynamics

Targeted Molecular Dynamics begins with two known conformations of a molecular structure, an 'initial' structure and a 'target' structure. The method calculates distance constraints that are applied to 'drive' the model from the initial structure to the target

structure in a molecular dynamics simulation. TMD enables the investigation of pathways that can involve fairly elaborate conformational changes. In some cases, the transition from the initial to the target structure may encounter too many barriers to be overcome directly. When this occurs, such differences can motivate the definition of intermediate structures to be added to the pathway (Kruger et al., 2001). The work by Kruger et al. developed such intermediates in TMD investigations of a large conformational change that occurs between the latent and active forms of Plasminogen Activator Inhibitor type 1, a protein that plays a role in blood clotting. The definition of the intermediate structure in this case was supported by the ability to manually define a rigid body rotation of the Reactive Center Loop (RCL), which is central to the functional activity of the protein. One concern that arises with the use of TMD is whether the application of external constraints improperly biases the potential energy surface of the structure so that physically unrealistic results are produced. (Schlitter et al., 2001) addresses this concern with the consideration that at room temperature, the fine structure of the potential energy can be neglected in comparison to the activated states that actually determine the rates of conformational change.

6.6.2. Steered Molecular Dynamics

The term Steered Molecular Dynamics (SMD) first appeared subsequent to descriptions of Targeted Molecular Dynamics (Izrailev et al., 1998). Steered Molecular Dynamics is a broader definition for the application of external forces in molecular dynamics simulations. The initial idea was inspired by application of forces in Atomic Force Microscope (AFM) experiments (Isralewitz et al., 2001a). SMD methods include forces

that rapidly produce large conformational changes, even at the expense of doing irreversible work and causing the system to depart from equilibrium. Such consequences are justified on the basis of qualitative insights that may be obtained, as well as quantitative information available from averaging ensembles of irreversible trajectories (Isralewitz et al., 2001b). Applications of SMD include use of a harmonic force to induce the rupture of biotin from avidin (Izrailev et al., 1997), the unfolding of the Ig1 domain of the cardiac muscle protein titin (Gao et al., 2002) and forced conformational transitions in 1,6-linked polysaccharides (Lee et al., 2004). These docking, unfolding and conformational applications attest to the variety of interactions opened up by the SMD approach.

6.6.3. Interactive Molecular Dynamics

Interactive Molecular Dynamics (IMD) is an interesting specialization of SMD methods. IMD refers to the application of external forces as defined by the interactive input of a user during a molecular dynamics simulation (Grayson et al., 2003). The use of computer graphics and haptic devices can be used to communicate the state of the simulation to the user. The feedback from the user to the simulation is typically sent from an input device such as the mouse or haptic controller. One significant constraint on the approach is that the simulation must be efficient and fast enough to make user interaction feasible.

In the work by Grayson et al, the required efficiency was obtained by simulating small subsets of the model, in which only about 1000 atoms were allowed to move. This work involved the membrane protein GlpF, which is an aquaporin used for water transport across the cell membrane. The interactive input consisted in dragging a small sugar

molecule such as ribitol or arabitol through the aquaporin channel and observing the resulting interactions with the channel and water molecules. The duration of the experiments were approximately 1 hour of human time, corresponding to 100-160 ps of simulated time. The results provided insight into the mechanisms necessary for transport of the sugars through the channel. These included the observation that the sugars underwent conformational changes at various locations in order to maximize the number of hydrogen bond interactions with their surroundings.

IMD is a promising approach because it enables the intuition and manual dexterity of the investigator to be more directly applied to the molecular world. Although the technique is currently limited by efficiency requirements, subsequent improvements in raw simulation power may allow researchers to come to grips with a much larger number of models. The many qualitative and quantitative results from all the methods above demonstrate that use of external forces and other information can be a successful strategy for getting around the timestep problem, and for investigating long timescale behaviors.

Chapter 7

Experimental Methods

"we have produced an atomic structure of the H. marismortui 50S ribosomal [subunit]."

(Ban et al., 2000)

7.1. Introduction

A large number of different experimental methods and affiliated analytical techniques exist for investigating biomolecular structure and function. They range from structural probing experiments with chemical nucleases (Bowen et al., 2001) to phylogenetic analyses of RNA secondary structure (Gutell et al., 2002). The main sources of data used in this dissertation were obtained from experimental techniques that produce detailed three-dimensional structural models of biomolecules. Consequently, the background will focus on these types of techniques.

For macromolecular structural modeling, the three most important experimental techniques available at this time are X-Ray Crystallography (XRAY), Nuclear Magnetic Resonance (NMR) and Cryo-Electron Microscopy (CryoEM). Each of these methods has particular advantages and limitations with respect to the size and resolution of the models that can be produced. X-ray crystallography is best at producing large models with high resolution. NMR provides good resolution as well as dynamic information for small models. CryoEM is a lower resolution method that works well for large and very large models. The results from one method are often used in complementary ways to assist

model building for the other methods. The collective capabilities of all three methods have generated a wealth of biomolecular structural data in recent years. This success has been especially significant for ribosomal research.

The size of a model refers to the number of atoms within it and is commonly expressed by molecular weight, with units in kilo-Daltons (kD). Small structural models can be characterized by sizes of 30 kD or less, which is in the size range of small proteins. Large size models may be 500 kD or more, and include the small and large ribosomal subunits.

The resolution of a model refers to the smallest features that can be resolved within it, and is commonly defined in Angstroms (A), where one 1 A is 10^{-10} m. The highest resolutions currently obtainable are in the fractions of angstroms, with the 0.54 A model of crambin as a notable example (Jelsch et al., 2000). This degree of resolution is more than enough to identify the positions of all of the atoms within a model, including hydrogens. Consequently, a model with this resolution is referred to as an 'atomic resolution' model. However, the term is also used for models of slightly less resolution in which all the atoms except the hydrogens are identified. For many biomolecular models of less than atomic resolution, the approximate positions of residues and chain backbones can still be determined, allowing for the identification of secondary structural motifs. Primary sequence data is often used in helping to make such approximations.

7.2. X-Ray Crystallography

X-ray crystallography is one of the most powerful experimental methods for producing structural molecular models. The method accounts for the largest number of structures submitted to the Protein Data Bank, with about 25,000 out of a total 30,000 structures

produced by early 2005 (PdbBeta site, 2005). It is capable of very high resolution, and resolutions in the range of 1.5 to 3 A are quite common. The recent success in producing x-ray crystal structures of complete ribosomal subunits, with model sizes of 50,000 atoms and more, indicates that the method is increasingly able to deal with very large, complex and asymmetric structures.

The steps needed to produce structural models from x-ray crystallography are broadly outlined as follows. First, the molecule of interest must be purified in a highly concentrated solution and then induced to form crystals suitable for diffraction. This is one of the most difficult steps in the process and it cannot be guaranteed to succeed for every desired target molecule. The target molecules must pack together in a uniform way to produce a regular, repeating 3-dimensional array of identical unit cells. Next, once crystals have been obtained, they are subjected to a high-intensity beam of x-rays to produce a diffraction pattern that is recorded using sensitive detectors. Then the diffraction pattern is mathematically analyzed to extract a three dimensional distribution of the electron density of the unit cell of the crystal structure.

One of the most difficult aspects of this analysis is solving for the phases of the diffraction waves. The measured data at each point in an x-ray diffraction pattern is actually a summation of radiation waves reflected from multiple atoms in the crystal unit cell. The phase problem is concerned with analysing the composite measured values and trying to determine from them the individual contributions from each atom (van Holde et al., 1998). If the phase of the reflections from individual atoms can be determined, then the measured intensities and phases imply a particular location for each atom. For larger structures, the phase problem becomes progressively more difficult. A number of special

techniques and advancements were required to determine the phases for the very large ribosomal subunits. These include the use of bright, tunable synchrotron radiation sources, special crystallization solutions, multiple isomorphous replacement with heavy atoms such as osmium, and anomalous scattering analysis (Wimberly et al., 2000), (Clemons et al., 2001).

X-ray crystallography does have some important limitations. The crystallization step has been one of the greatest bottlenecks in the process and until recently was considered not only more of an art than science, but an actual 'black art'. However, the terrifying laboratory supplications have slowly been giving way in recent years to more systematic methods involving automation, miniaturization and analysis of crystallization phase diagrams and kinetics (Chayen, 2004). Another potential issue for biomolecules concerns the conditions under which crystals are formed. Such conditions are often far removed from the physiological state. As a result, high concentrations and crystal packing effects may produce structural distortions or altered conformations that do not occur in vivo. For ribosomal subunit structures, these constraints have affected the resulting models in a number of ways. Most significantly perhaps is that the primary species for ribosomal research, *E. coli*, has not been amenable to high-resolution crystallographic analysis. Initial success only came about through the use of extremophiles such as *Haloarcula marismortui* (Ban et al., 2000) and *Thermus thermophilus* (Schluenzen et al., 2000), (Wimberly et al., 2000), which are bacteria that exist under greatly elevated conditions of salt concentration and temperature. Although the sequences of ribosomal genes between different species have been largely conserved, sequence differences do exist between the extremophiles above and *E. coli*, which has been the reference species for much of the

existing ribosomal research. Structural and functional variations that result from these species differences is a potential concern in the analysis based on the current crystal structure models of the ribosomal subunits.

7.3. NMR Spectroscopy

Nuclear Magnetic Resonance spectroscopy (NMR) is another important method for investigating biomolecular structure. Of the nearly 4500 non-crystal structures in the Protein Data Bank, NMR accounts for about 4400 of them. However, the most important limitation of NMR is that currently it can only produce models for smaller structures, generally 30 kD or less (Wider, 2000). This limitation is offset by the additional dynamic information that can be obtained with NMR, such as the multimodal distribution of methyl sidechain rotations in proteins (Wand, 2001). Another advantage is that NMR measurements are made in solution and are much closer to physiological conditions than the crystal samples of x-ray crystallography.

The principle of NMR derives from the non-zero magnetic dipole moments associated with the spin states of atomic nuclei. Only nuclei with odd atomic mass numbers have non-zero magnetic moments. This includes hydrogen, which provides the most important and ubiquitous signal for biological molecules. For other important biological nuclei such as carbon and nitrogen, which have naturally-occurring even atomic mass numbers, isotopes such as ^{13}C and ^{15}N can be used. The method works by placing sample molecules in a strong external magnetic field. This strong field causes a spreading between the energy levels of the magnetic spin states of the atomic nuclei in the sample, allowing the

naturally weak differences in spin state populations to be greatly amplified for detection. After the nuclei moments have been affected by the strong external magnetic field, an external electromagnetic field of variable frequency is applied to probe for shifts in the spin state energy levels. These shifts are affected by the local environment of the atoms, which consists of the surrounding electron clouds that perturb the local magnetic moments. The result is a series of shifted amplitude peaks in a frequency spectrum. Each peak corresponds to the signal from a particular atom type. Analysis of the chemical shifts and the correlations between them produce distance constraints and other information used in model building. For biomolecules, model building begins with structures determined from primary sequence and then refined with use of the constraints in distance geometry algorithms and/or restrained molecular dynamics simulations (Wider, 2000).

A nice example of NMR analysis of ribosomal structures is the solution structure of the A-Loop of the 23S ribosomal RNA (Blanchard and Puglisi, 2001). This work attempted to investigate the effects of 2'-O-methylation of residue U2552, which is part of a non-canonical pair at the base of a 5-residue loop. The A-loop contributes to recognition of the incoming aminoacyl-tRNA in the accommodation step of decoding, just prior to peptide transfer. In contrast with the single structures derived from x-ray crystallography, the restrained molecular dynamics simulations in NMR refinement typically produce an ensemble of energy-minimized structures. In the A-loop analysis, an ensemble of 15 structures were produced for the unmodified loop and 21 structures were produced for the modified version. The NMR structures differed significantly from the crystal structure. The results showed dynamic features such as the syn-anti conformational switching of

G2553. pKa shifts and resulting protonation of the N3 atom of C2556 were also observed, and were coupled to conformational changes within the loop. This kind of data is especially valuable for deciphering the peptide transfer mechanism, as pKa shifts in other ribosomal residues have also been shown to affect the process (Muth et al., 2000), (Katunin et al., 2002).

7.4. Cryo-Electron Microscopy

Cryo-Electron microscopy is a lower-resolution method for producing structural molecular models. The limit of resolution depends on the particular type of procedure and molecular sample, and can be as good as 4 A (Miyazawa et al., 2003). The higher resolutions are only obtained in fairly limited cases so far, and medium resolutions in the range of 15 to 7 A are more typical. This is still above the 3 A limit generally needed for identification of macromolecular folds. The technique derives from traditional electron microscopy and relies upon vitrification and cryonic suspension of the sample in order to protect it from damage by dehydration and electron radiation. (Baumeister and Steven, 2000) present a nice review of the basic principles of recent CryoEM methods as described below.

CryoEM can be subdivided into three different subtypes: 1) electron crystallography, 2) single particle analysis and 3) tomography. Electron crystallography creates diffraction patterns from samples where the molecules are in an ordered array. The analysis is similar to x-ray methods, and the method works on thin layers that would not be suitable for x-ray crystallography. Single particle analysis is a more flexible type of CryoEM

where the sample molecules can exist in random orientations, but the particles must be above a certain size threshold of around 300 kDa. The method works by starting with some kind of initial low resolution 3D model, which is then refined using the 2D images of thousands of individual particles. Electron tomography is the third and most general type of method. It can be applied to any type of molecule, organelle, whole cell or tissue structure. It involves a sequence of irradiations of a single specimen at multiple angles. This poses more difficult technical requirements for controlling the proper orientation of the sample and avoiding excess radiation damage. The advantages of the method are quite exciting however, as it holds the promise of producing structural models that span the microscopic and macroscopic worlds. An example is the given by the work of (Medalia et al., 2002), which shows the detailed structure of the actin cytoskeleton network of Dictyostelium cells in extraordinary and fascinating detail. As more structures like this are produced, they will likely prove to be fertile ground for theoretical analysis and multiscale modeling projects.

The use of CryoEM techniques has been very important for ribosomal structural modeling. Low resolution CryoEM data was used in x-ray phase determination (Cate et al., 1999) and also provided surface envelopes into which higher resolution structural models could be docked (Klaholz et al., 2003). In addition, CryoEM methods have produced low resolution structures of intermediate states in the translation process, involving conformations and components of the ribosome that are not available in x-ray crystal structures (Frank, 2003).

One unfortunate limitation for current CryoEM research is the lack of standards for data analysis and deposition. X-Ray and NMR structures are standardized on the PDB format

of the Protein Data Bank (Berman et al., 2000), which provides a well-defined representation of atom positions. This standard, in conjunction with public availability of the structural data via the Protein Data Bank, has been extremely valuable because it allows published structural models to be easily accessed by independent researchers. The standards and public access have also accelerated the development of analytical methods and software tools that make better use of the data. Standardization and availability of CryoEM data, although desirable, has been more problematic. There are currently multiple existing image and map file formats and only a few CryoEM models in the Protein Data Bank. Recently the CCP4 format has been proposed as a standard (Fuller, 2003) and a public repository for EM data has been created at the European Bioinformatics Institute (Em data website, 2005). A rigorous policy of public deposition of structural EM data associated with published models would naturally promote the adoption of common standards. The resulting standardization and public availability would no doubt greatly advance the practice and benefits of CryoEM research.

Chapter 8

RNA

"rRNA, probably in conjunction with ribosomal proteins, is primarily responsible for the function of the ribosome."

(Hill et al., 1990)

8.1. Introduction

After the earliest understandings of Ribonucleic Acid (RNA) emerged more than 50 years ago, there have been dramatic additional discoveries of its capabilities and functions in the living cell. These include its central function as an informational carrier in the form of mRNA (Moll et al., 2004), its structural role as the framework of the ribosome (Ban et al., 2000), the catalytic activities of ribozymes (Doudna and Cech, 2002b), and recently there have been very preliminary indications that it may possess a new kind of genetic capacity (Lolle et al., 2005). One begins to wonder if there is anything that RNA does *not* do.

8.2. RNA Structure

RNA is a nucleic acid, a property it shares with its close relation DNA. RNA is a polymer or chain of individual residues that are covalently bonded together. The structure of each RNA residue, also referred to as a nucleotide, can be further broken down into three component parts: an aromatic flat base, a ribose sugar and a phosphate group. The base

portion of the residue determines its identity and in RNA there are four types of bases commonly found: adenine, guanine, cytosine and uracil (abbreviated A, G, C and U). This contrasts with the 20 types of amino acids found in proteins. The structure and connectivity of the RNA residue components can be seen in Figure 1. The ribose is in the center with its pentagonal ring. It is connected on the right to the hexagonal ring of the cytosine base. On the left is the phosphate group, which consists of a phosphorus atom bonded to four oxygen atoms in a tetrahedral configuration. The residues of an RNA chain link together in a non-symmetrical way that gives the resulting chain an ordered direction. The 3' ribose oxygen of one RNA residue connects via the phosphorus atom to the 5' oxygen of the next residue in the sequence. A sequence is written in what is called the 5' to 3' direction, so the sequence 'AUG' means that an adenine at the 5' end of a three-residue chain connects to a uracil which in turn connects to a guanine at the 3' end of the chain.

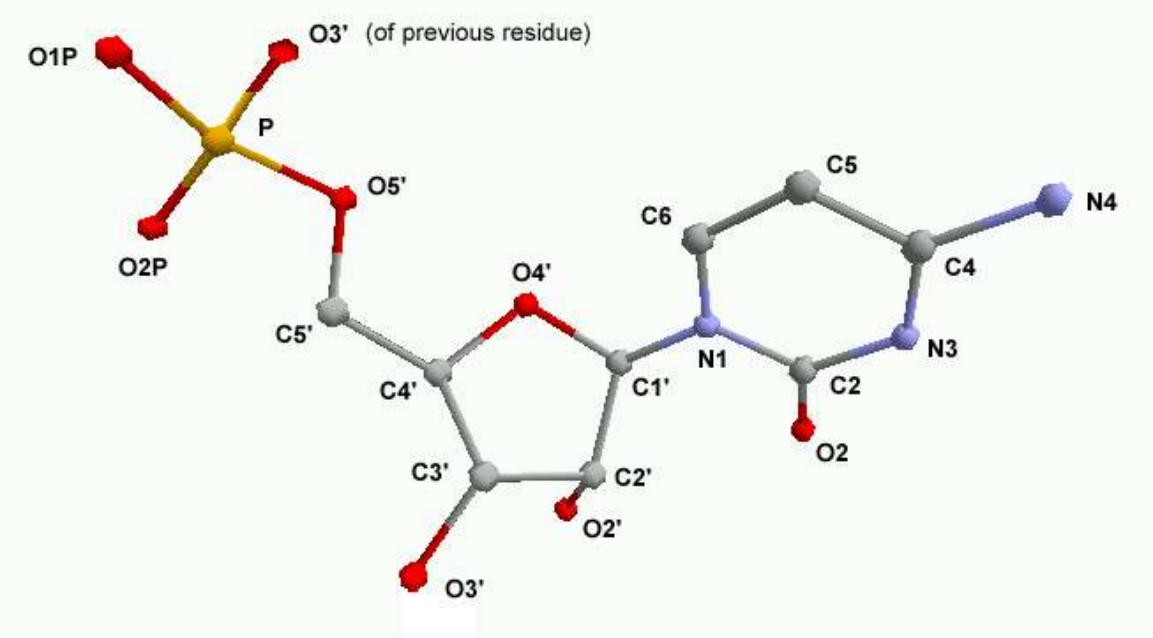

Fig. 1. Structure and atom numbering of a cytosine residue of RNA.

8.2.1. Secondary Structure

The specific sequence of bases in an RNA chain define its 'primary structure', also called 'primary sequence'. After RNA is synthesized as a linear chain, the specific interactions between the bases in its sequence cause it to fold up into localized shapes that compose its 'secondary structure'. The most important secondary structural feature for RNA is the helix, which is composed of a pair of antiparallel strands. The strands are held together by the formation of Watson-Crick base pairs between opposing residues. These base pairs consist of hydrogen bonds between the atoms in the flat rings of the bases. Watson-Crick base pairs, also called 'canonical base pairs', refer to the complementary pairing of an adenine base with uracil (A-U), and a similar pairing of guanine with cytosine (G-C). Stacking interactions between the aromatic bases above and below also contribute to the stability of the helix. Under physiological conditions, RNA forms a particular kind of helix called the 'A-form' helix, which contains a deep narrow major groove and a wide shallow minor groove, as can be seen in Figure 2. The hydrogen bonds between the Watson-Crick edges of helical bases pairs are by no means the only interactions that determine RNA structure. Hydrogen bond interactions can form between many of the other atoms in the residues and as a result RNA molecules can form a wide variety of secondary-structural shapes and discrete patterns called 'motifs'. Motifs are determined by a localized set of residue interactions and form distinctive structural patterns such as 'U-turns', 'A-platforms', bulges and tetraloops (Moore, 1999).

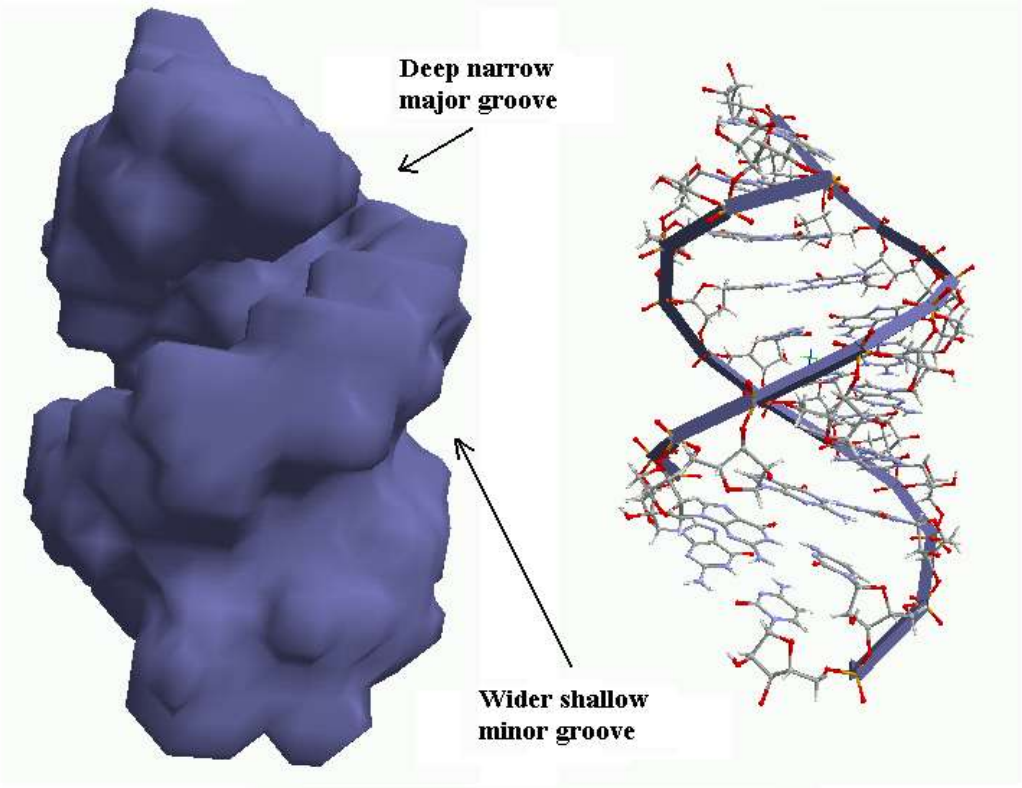

Fig. 2. Left: surface representation of an RNA helix in the A-form, showing the major and minor grooves. Right: same structure in backbone and covalent representation.

8.2.2. Tertiary Structure

From the interactions of the RNA secondary structural elements, an entire RNA molecule can fold up into a well-defined three-dimensional shape to produce what is called its 'tertiary structure'. An example of the tertiary structure of RNA can be seen in Figure 4, which is a representation of a tRNA molecule. Another important determinant of RNA structure is the negatively-charged phosphate backbone, with one negative charge accumulating per residue. The large negative charge of a long RNA chain must be

compensated somehow in solution through interaction with cations such as Na^{+} and Mg^{2+} or other positively charged molecules.

8.2.3. Geometrical Parameters

A number of geometrical parameters can be used to characterize RNA tertiary structure. The conformation of the ribose ring can be 'puckered' in various ways that are distortions from a planar shape. An important puckering parameter is determined by the positions of the C2' and C3' atoms relative to the plane defined by the C1'-O4'-C4' ribose atoms. The plane has two sides, endo and exo. The endo side, which is facing the viewer in Figure 1, is defined by the side containing the C5' atom and the base ring. If the C3' atom is displaced farthest from the plane toward the viewer, the pucker conformation is referred to as C3'-endo. If the sugar pucker 'flips', as occurs in transitions to other kinds of helices such as the B-form helix, the conformation would be C2'-endo. Another set of geometrical parameters for RNA are the six torsion angles α through ζ that are used to specify the conformation of the RNA backbone, as shown in Figure 3. Having six torsion angles per residue considerably complicates the analysis and classification of RNA backbone conformations, compared to the two backbone torsion angles φ and ψ in proteins. In addition to the backbone torsions, the glycosidic torsion angle χ specifies the orientation of the base ring with respect to the ribose. Bases can be in either the anti or syn conformation. The anti conformation is exclusively found for the pyrimidine bases C and U, where the bulky O2 atom of the base ring reduces steric clash by being on the exo side of the ribose. The purine bases A and G are also usually in the anti conformation but

can also be in the syn conformation for a narrow range of χ torsion angles (Blackburn and Gait, 1996).

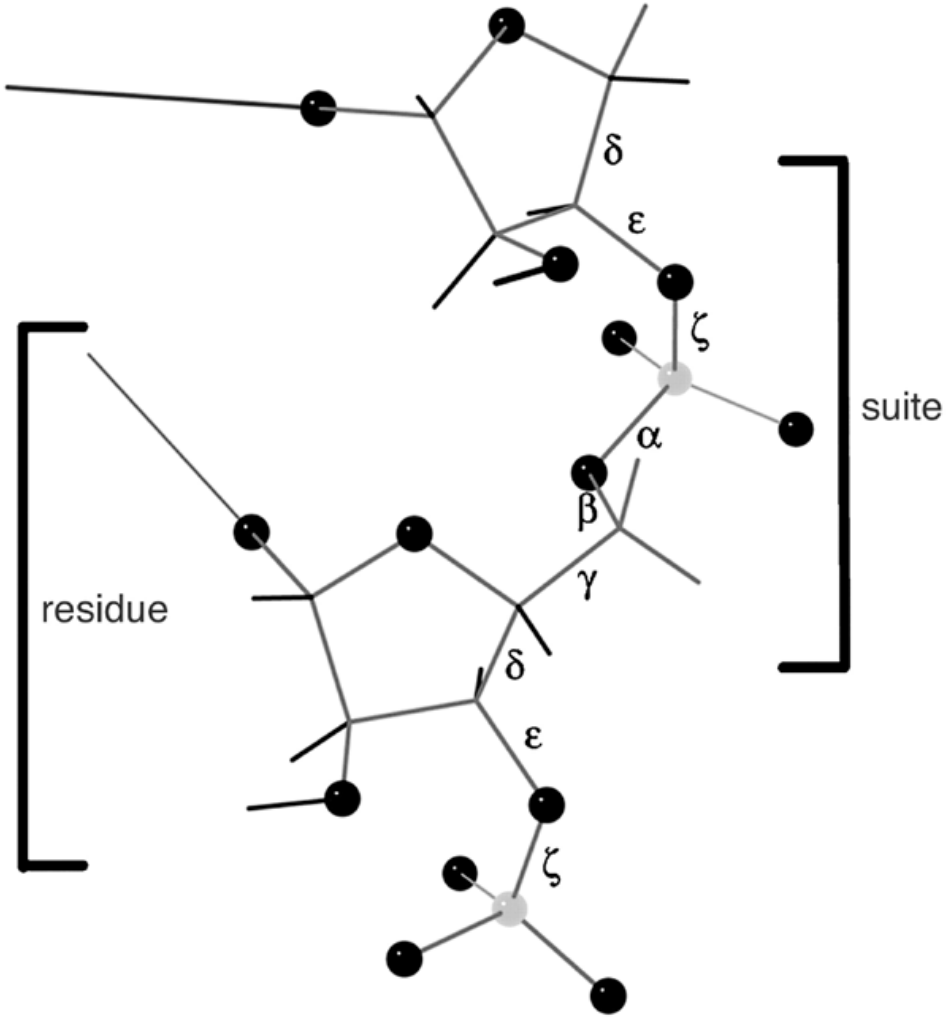

Fig. 3. Torsion angles in the RNA backbone. Each residue has 6 angles, α through ζ. The angles can also be organized across the linkage between two residues as a 'suite', consisting of the angles δ through γ. Figure is taken from (Murray et al., 2003).

8.3. RNA Function

The ability of RNA to form a variety of structural shapes is reflected in the many functional RNA molecules that are found in living cells. The different kinds of RNA

include messenger RNA (mRNA), transfer RNA (tRNA), ribosomal RNA (rRNA) and small nuclear RNA (snRNA) among others. mRNA represents a copy of a portion of the genetic information held in DNA. After an initial copy is made during the process of transcription, mRNAs are often subjected to considerable modification through the splicing together of various segments and removal of others from the initial message. At some point, the mRNA is delivered to a ribosome as a message for the creation of a new protein. tRNA and rRNA are types of RNA that are also used in the translational machinery of the ribosome, as discussed below. snRNA are bound together with proteins to form ribonucleoprotein particles in the nuclei of eukaryotic cells that participate in the processing of initial mRNA transcripts (Garrett and Grisham, 1999).

In addition to structural diversity, some RNA molecules have a catalytic ability to increase the rates of certain chemical reactions, a capability that they share with protein enzymes. Consequently, such catalytic RNAs are called 'ribozymes'. Examples include the self-splicing introns that are formed in pre-mRNA transcripts. These intron segments are able to induce their own removal from the adjacent exon segments that go on to form the ultimate mRNA message. Another example is the hammerhead ribozyme, a very small ribozyme that is used in viral replication (Doudna and Cech, 2002b).

Although RNA can perform structural and catalytic functions, it still retains an ability to be unfolded in a way that exposes its linear sequence of bases for copying and replication, a property that it shares with DNA. By possessing structural and catalytic capabilities on the one hand, and informational and replication capabilities on the other, RNA can perform functions that are currently done in modern cells with proteins and DNA. This has motivated the 'RNA World' hypothesis, which proposes that modern life evolved

from an earlier form of life that used RNA for both genetic and functional purposes (Joyce, 2002). This hypothesis provides a possible answer to the evolutionary paradox in which modern proteins are dependent for their existence upon DNA, and DNA are dependent for their existence upon proteins.

Chapter 9

The Ribosome

"An adaptor molecule could fit in only those places on the nucleic acid template where it could form the necessary hydrogen bonds to hold it in place. Sitting there, it would have carried its amino acid to just the right place where it was needed."

(Crick, 1988)

Introduction

To provide some background for the applications presented below, the main structural and functional elements of the ribosome are briefly covered here. For a more comprehensive introduction, a fairly recent overview of the ribosome presented by (Garrett et al., 2000) can be consulted. The focus in this research has been on bacterial ribosomes. Eukaryotic ribosomal machinery has some basic similarities, but is considerably more complex and elaborate in a number of ways, both structurally (Spahn et al., 2001) and functionally (Doudna and Rath, 2002a).

The entire system of translation of genetic information into functional products is complex and extends significantly before and after the immediate act of protein assembly on the ribosome. Post-transcriptional mRNA processing, the charging of tRNAs and regulatory activities can all be considered as important parts of the complete process of protein translation, but here the terms 'ribosome', 'ribosomal complex' and 'translational

system' refer to the immediate activities and components on the ribosomal complex that are concerned with assembling amino acids into a protein chain. The central phase of ribosomal translation is elongation, in which the protein chain is increased in length by one amino acid for each cycle of a repeating process. The elongation phase is preceded by an initiation phase that brings to components together in a regulated manner. Elongation comes to an end in an subsequent phase called termination. Both the initiation phase and the termination phase consist of many complex states and components that are distinct from the elongation phase, and they will not be covered here.

9.2. Ribosomal Structure

The main structural components for the elongation phase of translation consist of the two ribosomal subunits, a messenger RNA (mRNA) that defines the protein sequence, a collection of transfer RNAs (tRNAs) that provide the amino acids that will form the new protein, and two accessory components, EF-Tu and EF-G, that are called elongation factors. Atomic resolution structural models for all of these components now exist, and some of the significant structural features of the tRNA, the small and the large ribosomal subunits are presented below.

9.2.1. tRNA Structure

The tRNA was one of earliest functional RNA molecules to be structurally determined at atomic resolution. The model shown in Figure 4 was determined from an x-ray crystal structure by (Westhof and Sundaralingam, 1986).

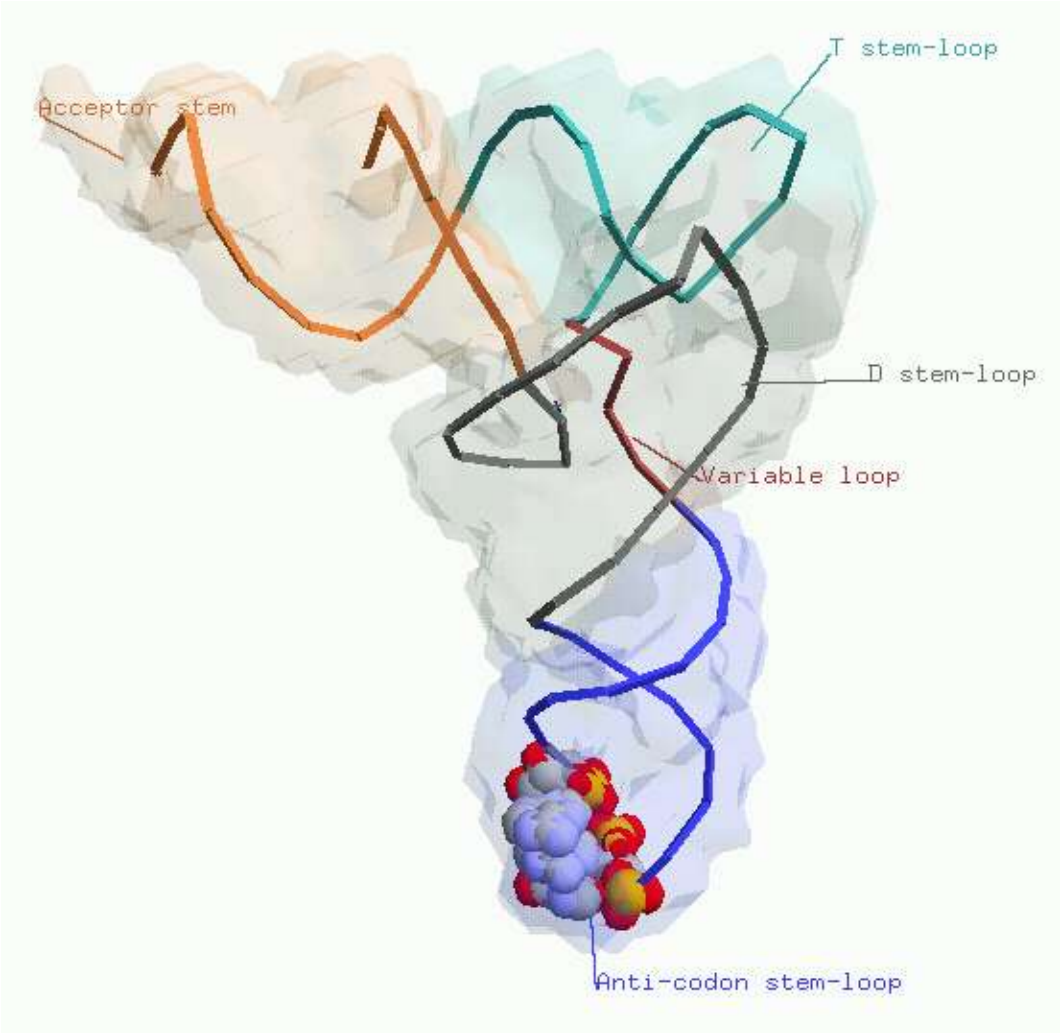

Fig. 4. Principal features of a tRNA molecule: Acceptor stem (orange), T stem-loop (cyan), D stem-loop (grey), Variable loop (red), Anti-codon stem loop (blue). The three anti-codon residues are shown in spacefilling representation. The structure is from PDB model 1TRA (Westhof and Sundaralingam, 1986), and the image is a screenshot from Ribosome Builder.

The tertiary structure has a distinctive 'L' shape that is formed by a number of A-form helical elements and stem-loops. The key parts relevant to translation are the anticodon stem loop (ASL), which contains the three nucleotides that base-pair with a codon on the mRNA, and the 3' acceptor stem, which holds the corresponding amino acid to be added to the protein. There are distinct tRNAs for each of the 20 amino acids found in proteins. Many tRNAs receive modifications to their component nucleotide bases, which in

conjunction with the nucleotide sequence acts to produce a set of distinctive structural patterns and chemical properties. Some of these properties facilitate proper recognition of the tRNA by the tRNA aminoacyl-synthetases that append the correct amino acid, and others properties promote recognition of the tRNA by certain ribosomal components during translation.

9.2.2. Ribosomal Subunit Structure

The two largest structural components of the ribosomal complex are the small and the large ribosomal subunits. The size of the subunits can be characterized by their sedimentation coefficient in Svedberg (S) units and for eubacteria such as *E. coli*, the size of the small subunit is 30S, the size of the large subunit is 50S and when they are associated together into a single complex the size is 70S. The small subunit consists of a single rRNA chain and some 20 proteins. The rRNA chain is referred to as 16S rRNA and contains about 1500 nucleotides. The large subunit consists of two rRNA chains and approximately 30 proteins. The large subunit rRNAs are referred to as 23S rRNA and 5S rRNA and contain about 2900 and 120 nucleotides respectively.

9.2.2.1. Secondary Structure

Long before atomic resolution tertiary models of the subunits became available, the primary sequence of the rRNA chains were obtained through reverse transcription and DNA sequencing, followed by determination of the RNA secondary structure through biochemical probing and mutational analysis. A secondary structure diagram of the 16S rRNA of the small subunit is shown in Figure 5. The structure is organized into 4 domains: the 5' domain in red, the central domain in green, the 3' major domain in

yellow, and the 3' minor domain in cyan. The domains roughly correspond to well-defined regions in the tertiary structure, with the 5' domain composing most of the body, the central domain forming the platform, the 3' major domain forming the head and neck, and the 3' minor domain defining a portion of the central interface to the 50S subunit (Wimberly et al., 2000).

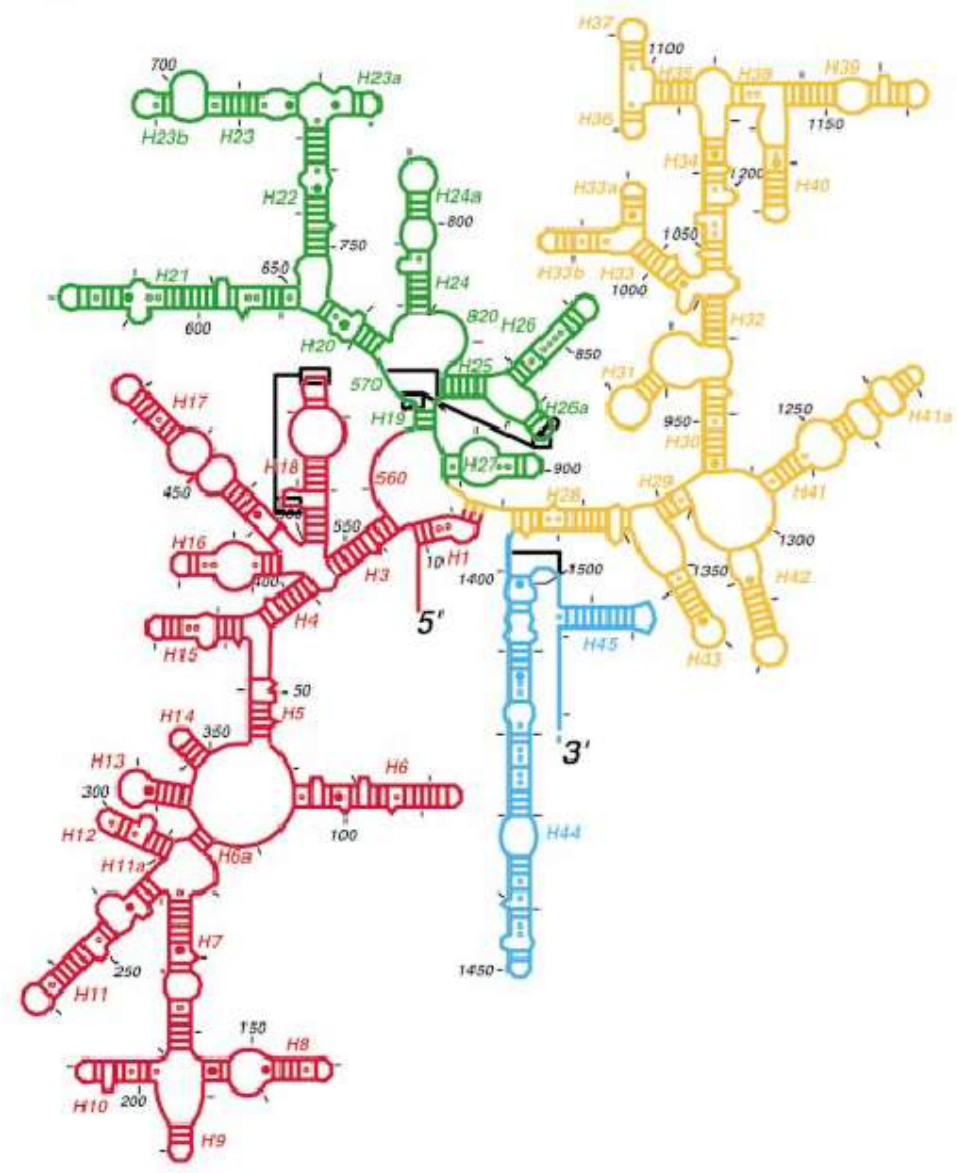

Fig. 5. Secondary structure of the 16S rRNA chain of the small subunit, with annotation of domains and helix numbers, from Fig. 2A, (Wimberly et al., 2000).

The secondary structure of the 23S rRNA in the large subunit is shown in Figure 6 and is organized into 6 domains. The correspondence of these domains with regions in the tertiary structure is less certain because the tertiary structure of the 23S is much more interconnected and compact (Ban et al., 2000). In addition to the domain boundaries, the structures of the ribosomal rRNAs are organized by a sequence of helix numbers which are used extensively as reference points in the literature for both the secondary and tertiary structural models.

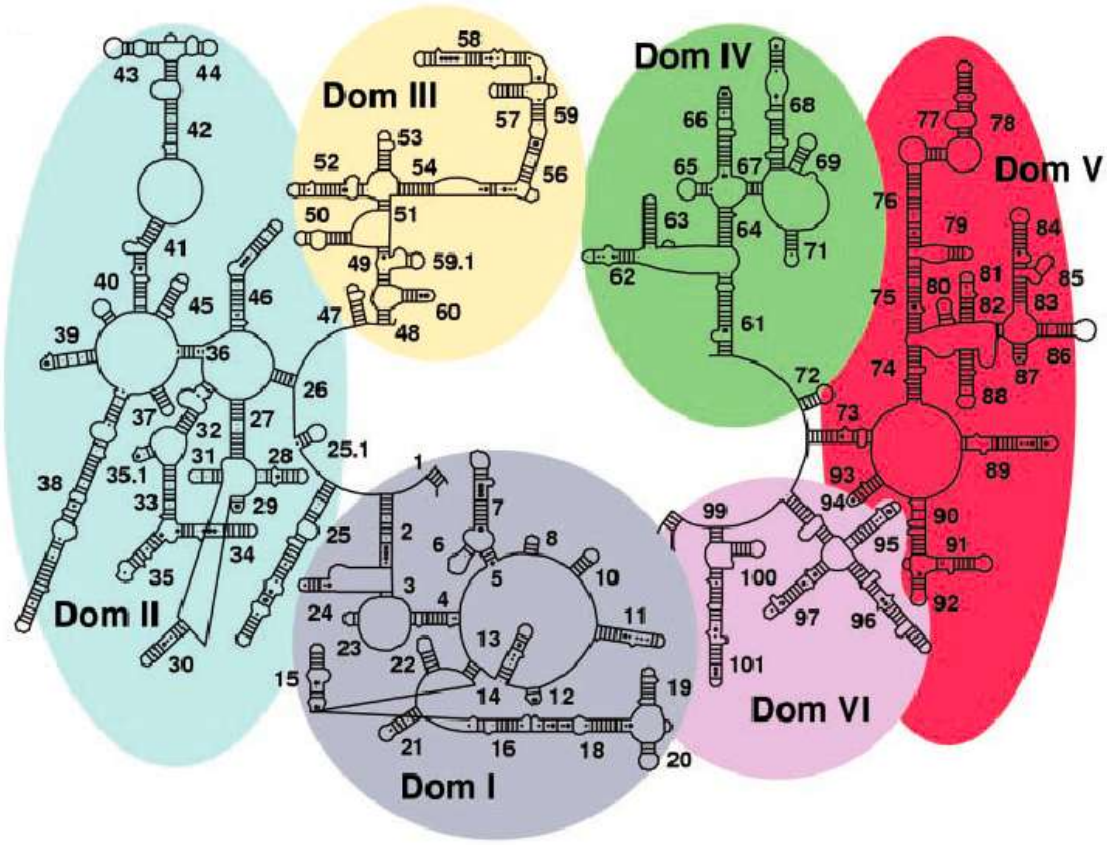

Fig. 6. Secondary structure of the 23S rRNA chain of the large subunit, with annotation of domains and helix numbers, from Fig. 4C, (Ban et al., 2000).

9.2.2.2. Tertiary Structure

The production of atomic resolution structures of the small and large subunits occurred in the year 2000 and represented an extraordinary achievement that was the result of many years of difficult work. The first atomic resolution structure presented was that of the 50S subunit of *Haloarcula marismortui* at 2.4 A resolution (Ban et al., 2000) and is shown in Figure 8. Shortly afterward, models of the 30S subunit of *Thermus thermophilus* were revealed including the one at 3 A resolution shown in Figure 7.

The general features of the 30S subunit are apparent in the domain organization mentioned above. As shown in the 'crown orientation', the head is the bulged structure at the top, supported by a flexible neck and connected to the larger body below. On the model to the right in Figure 7, the surface facing the viewer is called the 'interface' side because this side makes contact with the 50S during subunit association. The back side of the model (not shown) is correspondingly referred to as the 'solvent' side. In the interface or 'front' view, the platform is located on the right, between the head and body. The left-hand portion of the body just below the head is referred to as the 'shoulder'. In the front view, the triangular structure in the head pointing to the left is referred to as the 'beak' and the protruding helix at the bottom is called the 'spur'. When the mRNA is docked to the 30S during translation, it wraps in a path around the neck with its leading 5' end coming out to the right and its trailing 3' end to the left.

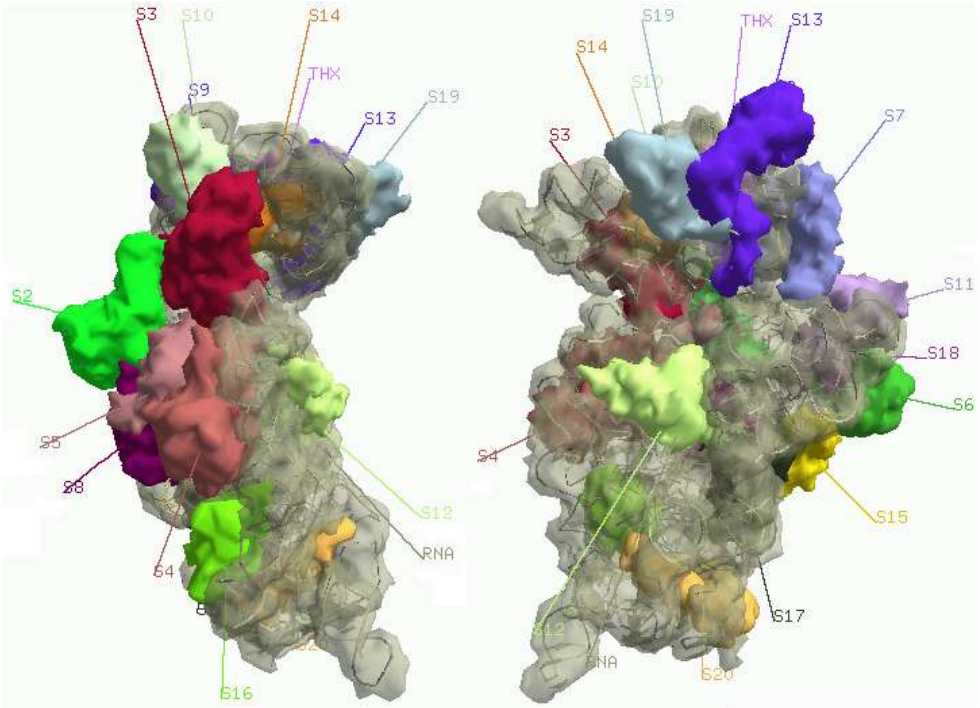

Fig. 7. Left side and interface views of the tertiary structure of the 30S subunit. The model is from PDB code 1J5E (Wimberly et al., 2000), and the image is a screenshot from the Ribosome Builder program.

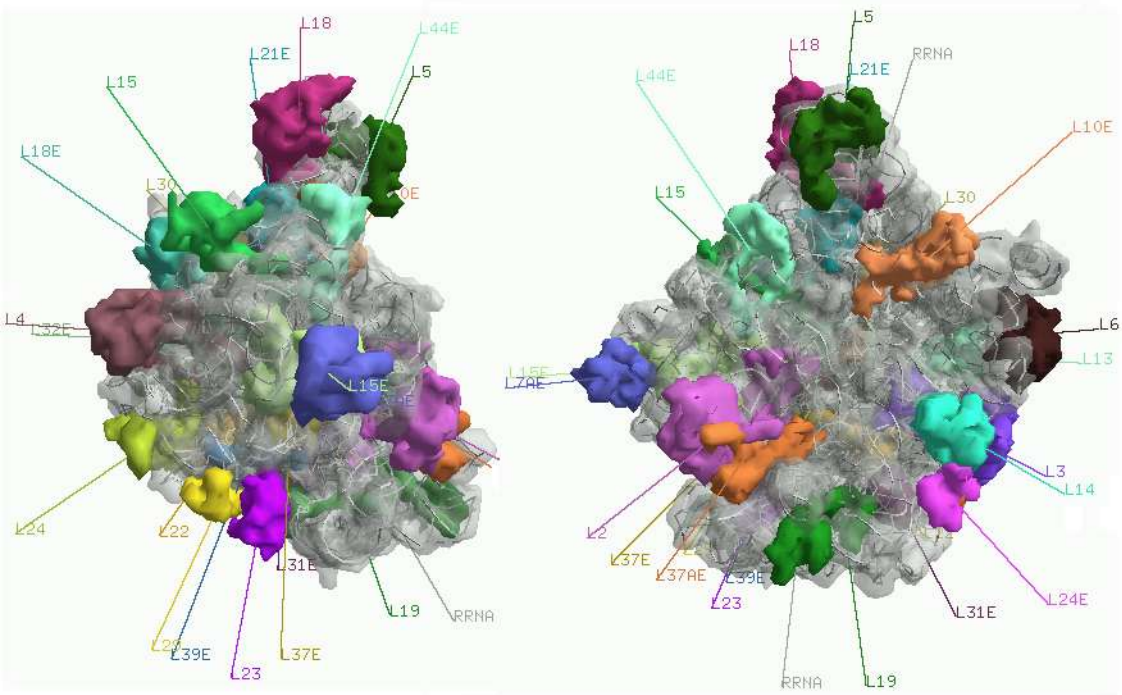

Fig. 8. Left side and interface views of the tertiary structure of the 50S subunit. The model is from PDB code 1FFK (Ban et al., 2000), and the image is a screenshot from the Ribosome Builder program.

The 50S structure in Figure 8 is identified with a different set of descriptive features. The bulge at the top is referred to as the 'central protuberance'. Not shown in the figure because they were not fully resolved in the crystal structure are two additional features. The first is the L7/L12 stalk, which would protrude up on the right near L10 and L6. The second feature is the L1 stalk, a similar elongated structure that would be pointing up on the left. The peptidyl transferase center (PTC) is the rRNA-only region located in the very center of the 50S subunit, where the amino acids are added to the nascent peptide chain. As the protein grows in length, it passes through a tunnel that goes from the PTC at an angle down and out through the back of the 50S subunit to the solvent side. Structural organizational features that are common to both the 30S and the 50S subunits can be observed in the large rRNAs that serve as structural frameworks upon which the proteins are subsequently bound. A number of the proteins are composed of globular domains connected to long tails that extend into the interior regions to act as structural 'glue' (Ban et al., 2000).

After these ground-breaking first models were released in 2000, the flood gates were opened to yield an abundance of additional atomic resolution models of complete subunits that were co-crystallized with additional components such as 1) antibiotics (Carter et al., 2000), (Pioletti et al., 2001), (Ogle et al., 2001); 2) translation factors (Carter et al., 2001), 3) mRNA (Yusupova et al., 2001), and 4) transition-state peptide transfer analogues (Nissen et al., 2000), (Hansen et al., 2002). An additional crystal model of both subunits was produced at the near-atomic resolution of 5.5 A to give a 70S structure (Yusupov et al., 2001), which has served as an important reference model upon which the higher-resolution models can be located.

9.3. Ribosomal Function

9.3.1. Elongation

During the elongation phase of translation, a sequence of amino acids are covalently bonded together to form a new protein chain. The amino acids are added one at a time, in a particular order that is determined by a corresponding sequence of nucleotides on an mRNA chain that is fed through the ribosome. An amino acid is added in each cycle, which is a set of specific actions that occur on the ribosome in a defined order. The set of actions that compose an elongation cycle can be divided into three sub-phases called the decoding phase, the peptide transfer phase and the elongation phase. An overview of the entire cycle is shown in Figure 9.

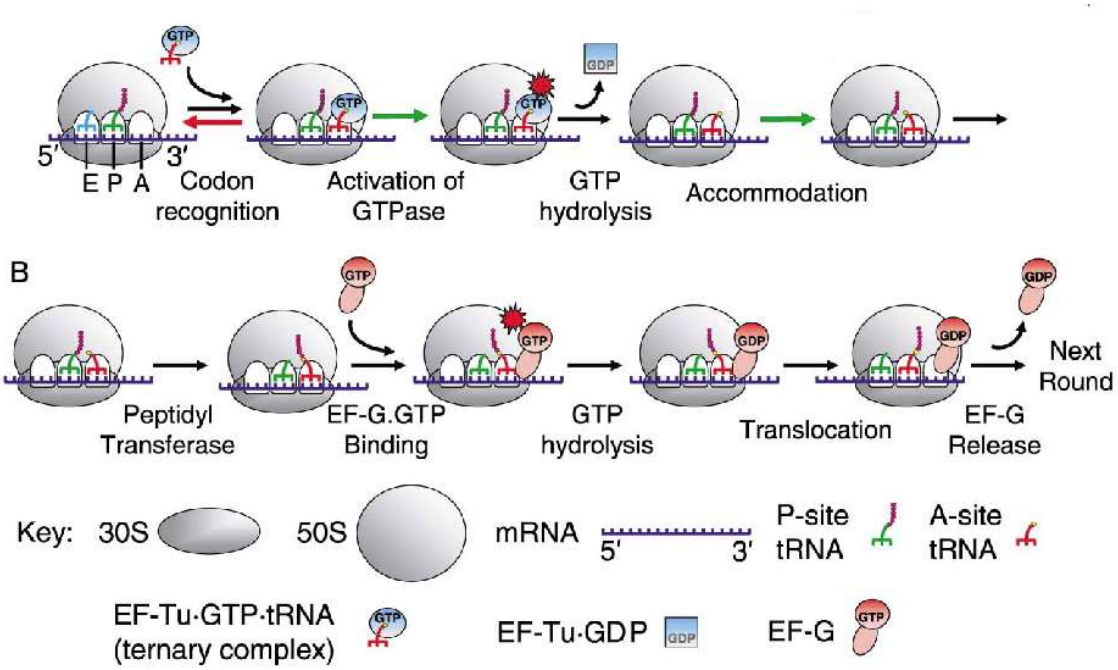

Fig. 9. Schematic overview of the elongation cycle of translation, from Fig. 3 of (Ramakrishnan, 2002).

9.3.1.1. Decoding

The decoding phase is the beginning of the elongation cycle. At this point, the small and large subunits have come together to form a 70S complex. The mRNA has been threaded in a path through the 30S subunit and a tRNA is located in the P-site, which is one of three well-defined locations for tRNAs as they pass through the ribosome. The three locations are called the 'A' site, the 'P' site and the 'E' site. The 'A' refers to the amino acid attached to the incoming aminoacyl-tRNA. The 'P' refers the growing peptide chain attached to the central peptidyl-tRNA. The 'E' refers to the exit site into which the P-site tRNA will move after peptide transfer. There are A, P and E sites on both the 30S and the 50S subunits. The amino acid and growing peptide are attached to the 3' arms of the tRNAs and are located in the A and P sites of the 50S subunit. The corresponding anticodon stem loops (ASLs) of the tRNAs are located in the A and P sites of the 30S subunit. The incoming A-site tRNA is actually delivered to the ribosome as part of a 'ternary complex' that also consists of the elongation factor EF-Tu and a GTP residue. The purpose of the decoding process of the ribosome is to recognize a tRNA that has the correct anticodon that corresponds to the current A-site codon of the mRNA. A codon is a sequence of three bases on the mRNA that will base pair with the complementary bases of a tRNA anticodon. If the correct codon is recognized, this will initiate the hydrolysis of the GTP residue bound to EF-Tu, resulting in the release of EF-Tu from the ribosome and allowing the incoming tRNA to fully enter the A site. The hydrolysis of GTP makes an energetic contribution to the process that increases the accuracy of recognition. At this point, the ribosome has entered the peptide transfer phase of the cycle.

9.3.1.2. Peptide Transfer

The process of peptide transfer is a chemical reaction in which the bond between the P-site tRNA and the nascent peptide is broken and the peptide is transferred over to the A-site tRNA, where a new bond is formed between the carboxyl end of the peptide and the alpha amino group of the incoming amino acid. The incoming amino acid remains bonded to the 3' end of the A-site tRNA, with the result of a deacylated tRNA in the P site and a peptidyl-tRNA in the A site that is longer by one amino acid. As the peptide grows in length, it begins to extend from the peptidyl transferase center (PTC) down through the tunnel in the 50S. The peptide transfer reaction is one that occurs spontaneously but very slowly by itself, and the ribosome acts as a catalyst to speed up the reaction rate. This is done by positioning the reacting arms of the tRNAs in a precise orientation to facilitate the reaction. It is possible that other kinds of catalytic contributions may also be made by the ribosome.

9.3.1.3. Translocation

After peptide bond transfer, the next and final phase of the elongation cycle is translocation. In this phase each of the tRNAs must move forward to the next binding site. The tRNA that was in the A-site must move into the P-site and the deacylated tRNA that was in the P-site must move into the E site. If a prior tRNA was in the E-site, it must dissociate from the ribosome. Along with tRNA translocation, the mRNA must shift by precisely three nucleotides in order to place the next codon for translation into the A-site on the 30S subunit. The total translocation involves very large conformational changes and movements over distances of 20 A or more. The ribosomal subunits are involved in

the process, but the precise details of their involvement as well as the timing and movements of the tRNAs and mRNA are not yet fully known. Translocation is catalyzed by the elongation factor EF-G, which binds to the ribosome during this phase and changes conformation in response to the hydrolysis of its bound GTP residue. Like EF-Tu, EF-G is a G-protein that consumes energy in order to assist the activity of the ribosome. In addition to the translocation movement, the translocation phase may include additional events that are needed to fully return the ribosome to a state where it is ready for the next cycle of elongation.

Chapter 10

Computer Software

"Computers were made for biology"

(Levitt, 2001)

10.1. Introduction

This section present some background information that is relevant to the software development of the Ribosome Builder program. This includes a brief survey of some existing scientific software used in molecular modeling, simulation and visualization. Then the subject of Open Source software is presented, along with its relevance for scientific software development. The chapter concludes with some reasons for the development of new software and a summary of the existing third-party software and tools that were incorporated into the Ribosome Builder program.

10.2. Scientific Software

The scientific software that is relevant to the ribosomal research in this work can be roughly divided into three categories: modeling software, simulation software and visualization software for structural molecular biology. Many actual software libraries and programs include features that overlap these categories, but the categories usefully reflect some natural and practical divisions of the problem domain. Structural molecular

modeling software is generally concerned with enabling a user to construct structural models, either de novo, or more typically, from experimental data such as X-Ray, NMR and cryoEM sources. Modeling software can also allow the user to view and modify existing structures. Simulation software generally includes molecular dynamics and other kinds of programs that can produce trajectories and dynamic changes in molecular structures over time. Visualization software, as its name implies, is concerned only with producing visual representations of molecular structures, both interactively and as high-quality renderings for publication.

XPLOR (XPLOR website, 2005) is a molecular modeling program that is often used for building the structural models obtained from x-ray crystallography. Molecular mechanics programs such as AMBER (Cornell et al., 1995) and CHARMM (MacKerell et al., 1998) can also be used in the modeling process to refine the starting hypothetical structures through energy minimization. The Molecular Modeling Tool Kit (MMTK) (Hinsen, 2000) is an example of a software library for molecular modeling. A library is a software development component that differs from a software program. A program can be directly executed by a user while a library provides pre-designed functions and resources that are subsequently incorporated into a standalone software program. The MMTK library is fully open source and includes a version of AMBER for energy minimization. MMTK is implemented in the Python programming language (Lutz, 2001), which is a scripting language that enables rapid prototyping and provides good support for integration with other software programs and tools.

Existing molecular dynamics software include GROMACS (Lindahl et al., 2001) and NAMD (Kale et al., 1999). GROMACS is a high-performance implementation that runs

on a variety of standard desktop systems. It also has a parallel capability using MPI, a standard software communication interface between processors (Foster, 1995). NAMD also has high performance parallel processing capabilities, with support for the new TeraGrid resource (Teragrid website, 2005), an open distributed computing infrastructure for scientific research . NAMD is also designed for integration with VMD (Humphrey et al., 1996), which is an associated program for visualization and Interactive Molecular Dynamics (IMD) (Grayson et al., 2003).

One of the pioneers of desktop molecular visualization software is the venerable RasMol program (Sayle and Milner-White, 1995), which has been the inspiration for a number of subsequent molecular modeling and molecular visualization programs including the Ribosome Builder. Although written to provide very good performance, it has now become somewhat dated, but its open source code lived on to some degree in subsequent programs such as CHIME (CHIME website, 2005), which was a plugin that allowed visualization in an HTML browser. Another descendant was Protein Explorer (Martz, 2002), which is also an HTML-browser based molecular visualization program. Both of these visualization resources are largely moribund now due to open source issues as discussed below. A more recent and powerful software visualization program is PyMOL (DeLano, 2002), which is an efficient implementation written in C (Darnell and Margolis, 1991) and Python, and allows for extension and customization of the core features. Other recent visualization programs are Cn3D (Cn3D website, 2005) and Jmol (Jmol website et al., 2005).

10.3. Open Source Software

Open Source software refers to the availability of the source code that is used in creating computer programs. Most existing software is created from human-readable information and then compiled into machine-readable binary forms. Access to the original human-readable 'source code' is generally required for practical analysis, modification and extension of a software program. Some helpful analogies exist between the 'source code/binary program' scheme in computer programs and the 'genotype/phenotype' distinction in biology. The issues surrounding open source software are both practical and philosophical. The practical issue relates to how complex software programs are initially created, debugged, used, maintained, extended and re-implemented. The most prominent example of complex software enabled by open source methods is probably the GNU/Linux collection of operating system software (GNU website and Stallman, 2005), although there are many other critical components of our modern software infrastructure that have been similarly dependent upon open source principles for their existence. A good introduction to the origins and history of open source development of GNU/Linux is found in (Moody, 2001). A more detailed and technical examination of open source practices is found at (CB website and Raymond, 2005).

The references above consider the importance of open source for the development of general kinds of software, but the issue is especially relevant for scientific software. In comparison to proprietary and commercial applications, scientific software is typically developed for new, experimental and uncertain areas of investigation. In addition, new scientific software usually builds upon a large existing formal mathematical and

theoretical infrastructure. These two qualities of exploratory research and logical cohesion are strongly supported by open source principals. The inability to understand and extend close source software inhibits exploratory research. The restrictions on the reuse of proprietary software prevents the integration and encapsulation of software into more functional forms. Examples of both of these kinds of restrictions can be found in existing scientific software. The release of the RasMol source code into the public domain enabled its reuse in an HTML-browser through the CHIME plugin. A certain degree of innovation resulted from the CHIME-based Protein Explorer, but subsequent development and use have largely stagnated due to the proprietary, closed-source nature of both CHIME and Microsoft Internet Explorer, upon which Protein Explorer is largely dependent. This pitfall of closed-source dependence can be found in other scientific software programs, but there are other problems as well. When software is closed-source, it can often disappear when the owning commercial entity changes hands or dissolves. One example is the TEXAS program, which was used for Quantum Mechanical calculations in the 1970s and 1980s (Pulay et al., 1979) and is now no longer available. As a result, it has become impossible or impractical to confirm or fully understand the scientific literature that used the TEXAS program. An example of an even more extreme problem that can occur is seen with the commercial program Gaussian, which has played a large part in QM research in the last two decades. The proprietary nature has enabled the corporate owners of the program to prohibit the use of Gaussian by certain scientific researchers and academic institutions (Giles, 2004), (BannedByGaussian website, 2005). In addition to this explicit relation between open source and scientific software, there are more general issues at stake, with larger political and philosophical implications. The

advance of science is critically reliant upon the open and free exchange of ideas. The recent explosion of information technology is transforming the domain of science. The traditional scientific domain has consisted of concepts, ideas, abstract algorithms and theories. With the emergence of the computer, additional aspects of science are entering this virtual domain. These aspects include strictly experimental data such as the human genome sequence, and research procedures such as software codes for molecular simulation. As the virtual domain of science expands, mechanisms such software patents, non-disclosure agreements and intellectual property laws represent a threat to the scientific process. The adoption of open source principles and practices will become increasingly important and necessary for the continued progress of science in the years ahead.

10.4. Software Creation and Re-Use

The requirements for adequate performance in working with models of ribosomal size was a major issue in the design of the Ribosome Builder software. At the time that the first subunit models became available, many of the visualization programs had difficulties handling models of this size. The model size also warranted against the use of existing molecular dynamics software. When used on the single-processor desktop computing resources that were available, the existing software would not come close to simulating the large models and long time scales that were desired. Also, for the reasons given above, the requirement for existing software to be open source was mandatory and a number of programs that might have been used as a starting point for the research were ruled out on this basis. Finally, the use of Steerable Molecular Dynamics (SMD) and

schematic annotations required new software features and capabilities, with the result that most of the Ribosome Builder software was created from scratch.

A number of third-party libraries and tools have been incorporated, however. The third party libraries that have been used include the Lua script interpreter, the Qt application framework, and some networking and geometry code. The use of third party tools includes a stripped down version of Perl (Wall et al., 1996) and Gnuplot (gnuplot website et al., 2005), which are used for creating and formatting some of the HTML reports for structural analysis. All of the software that has been used is non-proprietary and almost all of the source code is available, with the exception of the Qt framework in the Windows version of the program. This is expected to change with the upcoming release of Qt 4.0, and should be less of an issue in any event when the Ribosome Builder program is ported to the GNU/Linux platform, which is one of the top priorities in the subsequent development.

Part II: Methodology

Chapter 11

Implementation Overview

11.1. Introduction

This section describes the software implementation of the Ribosome Builder program. The core of the program consists of a molecular dynamics engine with the capability to define steerable molecular dynamics simulations. This engine and its associated forcefield were developed from scratch to enable the simulation of long timescale activity with the intent of investigating large macromolecular systems such as the ribosome. In order to meet such ambitious goals, it was necessary to construct a comprehensive and integrated set of components for molecular modeling and simulation. These components include 1) an efficient data representation, 2) algorithms for 3D geometry and chemical interaction, 3) a parametrized forcefield, 4) a steerable dynamics engine, 5) user interface, and 6) script interpreter and API.

To place the software implementation in a better perspective, the intended use of the program is first shown in an overview. A brief discussion of design considerations and lessons learned is given. This is followed by a summary of the project architecture and software components. The subsequent chapters in the methodology section then present and discuss the software components in detail.

11.2. Intended Use of the Application

An overview of the input, tools and output of the program is shown in Figure 10. The program can be used in conjunction with a web browser, as discussed in more detail below. The primary input data consists of atomic structural models in the form of PDB files from the Protein Data Bank (Berman et al., 2000). A number of ribosome-related PDB files are distributed with the program. The third input component consists of user-defined script files, which are used to automate and customize the behavior of the program.

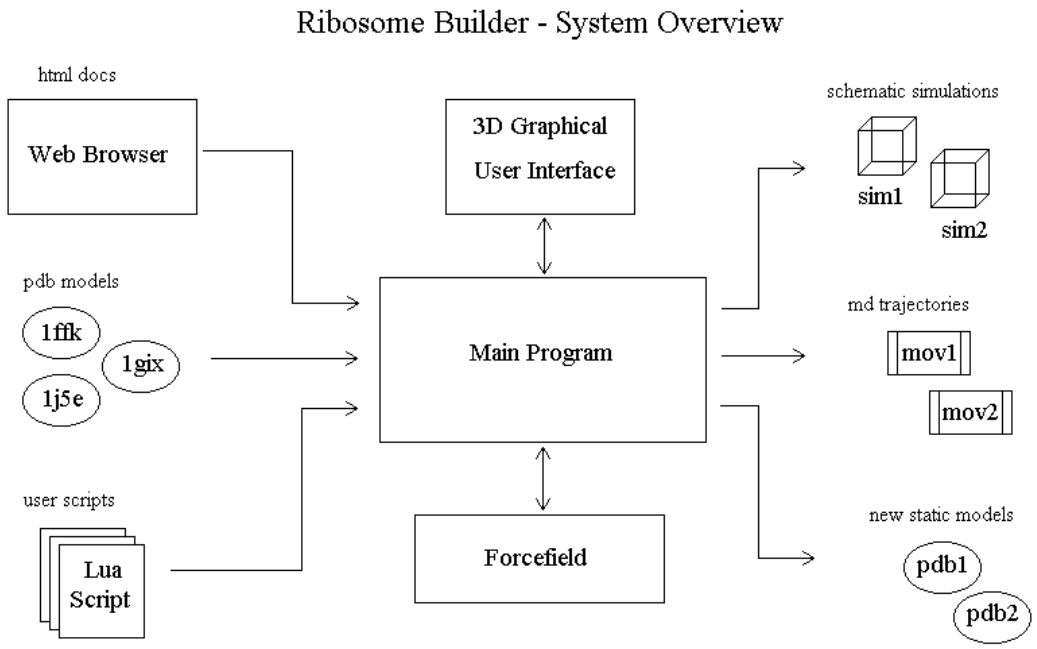

Fig. 10. Block diagram of the Ribosome Builder system.

The 3D graphical user interface and simplified forcefield are built-in components of the main program. The results of working with the program include new static structural models that have been produced from changes to the input models. Molecular dynamics trajectories can also be output as movie files, for subsequent playback. Both static models and movies are then typically incorporated as components, along with scripting code, to produce a schematic simulation, which can be viewed in the 3D graphics window. If a preliminary impression of the program's graphical user interface is desired, it can be examined below in section 14.1.

11.3. Some Design Considerations

To fulfill all of the requirements mentioned above, a fairly large and complex program was ultimately developed. As software programs grow in size and capability, the development and maintenance of the software can become quite challenging. Good software design practices are essential to prevent the code from collapsing into a disorganized and poorly-performing mess, or worse, a bug-ridden and mysteriously crashing nightmare. Such design practices include the use of object-oriented programming techniques, which attempt to separate the different functional aspects of the program into self-contained modules. Another valuable practice is the use of unit tests, which validate the code as it is written. The practice of unit testing was done to a limited degree in the project, and with some success. It is expected to be employed more heavily in future development.

Even with the use of such techniques however, the 'right' way to specify the program architecture is not always obvious at the beginning. Mistakes were made and the program went through several versions before a manageable, efficient and fully-developed architecture was achieved. Detailed discussion of some of these 'lessons learned' is done below in subsequent chapters, but two general design decisions are worth noting here. The first was to minimize the size of the data structure that held the raw atomic structural data. Bloated memory usage can cause some of the largest impacts on program performance, and the separation of optional data parameters from fundamental ones was an effective decision. One aspect in which this was done was the separation of static model data from dynamic model data. The second important lesson learned was to favor class composition over deep class hierarchies. This can be a common mistake in inexperienced object-oriented design, and the failure to adhere to that principle resulted in some serious headaches in earlier versions of the software.

11.4. Software Components

The source code that comprises the program software has been extensively modularized into a large number of separate units called 'libraries'. This collection of libraries, in conjunction with the main application code, is grouped into three larger categories: low-level libraries, application-specific libraries and the main application code. The low-level libraries contain code that is more general in purpose and can be re-used in other kinds of software applications. The application-specific libraries, as the name implies, are tied more closely to the main program, both in purpose and through include-file dependencies.

The low-level libraries include code for general-purpose mathematical and geometrical operations, basic representation of atomic and molecular data, component support for molecular dynamics forcefields, graphical operations, parsing, networking and miscellaneous utility functions. The application-specific libraries provide for representation of graphical annotations, scripting, and a generalized framework for navigating and selecting objects in a three-dimensional space. The main application modules include the specific graphical user-interface components needed for the program, document and high-level data classes, and the steerable molecular dynamics engine. The software classes for these core libraries and main application were all implemented in C++.

On top of the core C++ code, a second layer of components was developed in the Lua scripting language that is supported by the program. This includes an extensive set of functions to interface with the main application, in the form of rbq_lua functions, and a higher-level library of Lua functions in a variety of functional categories.

Chapter 12

Low-Level Libraries

12.1. wkChem Library

The primary purpose of the wkChem library is to provide a lightweight interface for atoms, bonds and fundamental chemical properties.

12.1.1. Motivation and History

One of the painful mistakes during an early phase of the project was designing an atom class that was both overweight and tightly embedded within an overly complex class hierarchy. The class was overweight in the sense that it tried to define data and operations for an atom object in many different contexts. These contexts included an atom as represented in a PDB data file, an item in a generic container, an atom used for static structural modeling, an atom within a dynamic forcefield and an atom drawn as a 3D graphical object.

The consequence of being overweight was that an atom would have member data for all of these different contexts. If a simple operation or calculation needed to be performed on a large set of atoms, such as calculating the total mass, a great deal of unnecessary memory was wasted in having to allocate the unused member data.

The complex class hierarchy also caused serious problems when trying to extend or reuse the existing code in different applications. The CAtom class was part of a class hierarchy inheriting from base classes CVoidPtrArray, CGraphicsObject, CForceObject and CBlock. Each of these classes defined large numbers of member functions. As a result, many of the derived class in the hierarchy would possess a huge number of member functions. Whenever a change was made to a member function in one of the base classes, this would often require a lot of changes in many locations in the code, where the overridden member would have to be re-implemented in the derived classes. In addition, these base classes would drag in a huge number of dependencies whenever a derived class, such as the CAtom class, was used in some small, independent application. The result was that much of the code became a huge, interlocked system which made reuse and rapid prototyping extremely difficult. A term has been coined for this phenomenon: the 'Big Ball of Mud' (Foote and Yoder, 1999), a metaphor that aptly expresses the errors and inevitable large project nightmares that result from bad software design practices. Another negative aspect of this older implementation was that data access was done using pointers to the actual object types. For example, say that some function was processing a residue object and needed to obtain the name and sequence position of that residue. In the old implementation, the function would receive a pointer to the residue object. The function would then cast the pointer to the actual specialized class type of the residue and then call its member functions to obtain the name and sequence number data. In doing this however, the specialized residue class is exposed to all parts of the application that need the data, with all the negative consequences resulting from such exposure (tight linkage between all parts, also leads to a big ball of mud).

So the new implementation was designed to provide access to objects in a hierarchy without using the specialized classes of objects. It did this in two ways. The first way uses index parameters. The second way uses interfaces. The use of index parameters is discussed below in the section on the main application. The development and use of interfaces in the wkChem library is discussed below.

12.1.2. CAtm

There are situations in which it is useful and desirable to use a pointer to access an object. To provide pointer access while avoiding the problems of specialized class exposure, a new generic atom class, CAtm, was defined. Because the class does not define any member data, it serves as an 'interface'. An interface is something that specifies a set of operations, which are usually organized around some common purpose or concept. The term interface is one that is formally defined in the Java programming language (Standish, 1997), but the equivalent construct is associated with the term 'abstract class' in the C++ programming language (Stroustrup, 1997). The difference between an interface and a class is that an interface cannot be instantiated. The interface serves only as a way of defining how something should be done. The implementation of this definition is then done by defining a concrete class, which can instantiate actual objects, each of which may contain member data.

The CAtm interface declares a minimal number of public member functions that were found to be necessary in many different areas of application. They include returning the element type, the spatial position, and an extended atom type, returned as an integer

value. In addition, the interface defines a default implementation for each of these functions, so that overriding them in derived classes is optional.

The extended atom type is an important property to have for atoms that may be a source of data for a molecular mechanics forcefields. Forcefields typically define a large variety of atom types that are much more specific than just the atomic element number (as discussed in further detail in the forcefield section).

For atoms that are read from PDB files, the more detailed atom type needed for use in a MM forcefield may or may not be determinable from the information in the PDB ATOM record. A useful and flexible feature of the atom type property, as currently implemented, is that in the absence of any specific information about the type of an atom, a default type can be returned which is simply equal to the element atomic number. The more specific atom type values are defined starting after the value of the final defined element, Mt. Since the atom type attribute does not contribute any overhead, has a good default value and can be useful as a descriptor of atom properties in numerous contexts other than molecular mechanics applications, the inclusion of this attribute in the minimal CAtm interface seems justified.

The CAtm class also provides definitions help to manage collections of CAtm objects, such as container types and an iterator class.

12.1.3. CBnd

Another fundamental chemical property is the covalent bond. This is a natural, pairwise association involving two atoms. As the potentially large number of atoms in an

application requires efficient implementation of data types and operations, this is equally true for covalent bonds, which are typically present in similar amounts.

There are a number of different ways to represent collections of covalent bonds, and the appropriate representation depends on how the bonds are used. One of the simplest ways is to define a covalent bond as a pair of pointers to CAtm objects. This is the definition used for the CBnd class. Then, for a collection of atoms, the set of all bonds for those atoms is a set of atom pointer pairs, defined as an array of CBnd objects.

However, for certain operations, such a representation can incur a large performance penalty. One example is a function that must return all the atoms that are bond partners of some parameter atom. If the bond representation is a list of CBnd objects, the function must iterate the entire list of atom pointer pairs for every param atom, with a growth rate of O(N). For time-critical tasks, such as computing steric interactions, this would produce a significant performance bottleneck.

In such situations, a different representation of bond collections is required. If a set of atoms can be represented as a numerical sequence, then bonds between atoms in that set can be represented as pairs of integers, where each integer is an index into the atom sequence. Then, a one-time O(N) operation can be done to build a array of integers for each atom, where each integer is the index to a bond partner atom. The set of all integer arrays then represents all the bonds in a form that enables rapid O(1) lookup of bond partners for each atom.

The CBnd class defines data types and operations to support this second type of bond representation. In addition, because a set of integer arrays can be awkward to use as a

parameter and return type, the class also provides several packing and unpacking functions that interconvert this list of integer lists into a single flat array of integers.

12.1.4. CElement

The CElement class defines a table that hold some basic properties for the first 108 elements, such as element symbol, name, mass, van der Waals radius and standard valence. The original source of the table was from the RasMol source code (Sayle and Milner-White, 1995), with modification of some of the values from (Schlecht, 1998). This class also currently defines these same properties for the extended atom types, which are used as parameters in the forcefield. Unlike the element properties, there are several member functions for modifying the atom type values, which allows for run-time customization of atom properties in the forcefield.

12.2. wkFastPdb Library

The purpose of the wkFastPdb library is to rapidly and efficiently read data from Protein Data Bank files and to make that data available to other software components.

12.2.1. Parsing PDB files

One of the principal goals of the program is to allow the modeler to quickly and conveniently insert and remove multiple models, in an interactive manner, during the modeling process. In order to meet this goal, the program was required to rapidly read the data from very large PDB files. Atomic-resolution models of each of the subunits of the

ribosome contain coordinates on the order of 50,000 atoms. This requires reading and parsing data from a corresponding number of ATOM record lines in a file.

This requirement was accomplished by defining a simple, minimal data structure that retained only the most necessary data, largely in its original form, and deferring more complex processing and representation to other parts of the program. The static representation of an ATOM record contains just the atom spatial position, its PDB serial number, its PDB name, HETATOM flag, and just a single additional integer variable, iAtomType, which allows for subsequent specialization of the atom as a particular chemical type.

In addition to the speed advantages that result from such a minimal representation, the PDB parsing routines avoid a number of problems that can result from inconsistencies in the format of a PDB file. While there is a very clearly documented specification of the PDB file format (RCSB, 2004), the format has changed over the years and there are many data files that are written in older or slightly non-standard versions. Numerous small differences and variations can be expected to be encountered when parsing a PDB file, reflecting the natural diversity of biological molecules and experimental conditions.

When reading a PDB file, the RB program requires only ATOM records to be read. There are many additional types of records which provide information about the model, and some of these records are read if found, but they are not required.

Another advantage of this minimal approach is that it becomes possible to 'cut out' a section of ATOM records from a PDB file and save it as a new file, which the program can then read as a completely independent fragment of molecular structure. This facilitates analysis, model building and interoperability of computational tools.

12.2.2. Classes

The structural relationships of the class components of the wkFastPdb library are shown in the UML diagram of Figure 11. The library consists of four classes: CFastPdbObj, CFpoChain, CFpoRes and CFpoAtom. CFastPdbObj provides the public interface to outside code and the other three classes are used for internal data representation.

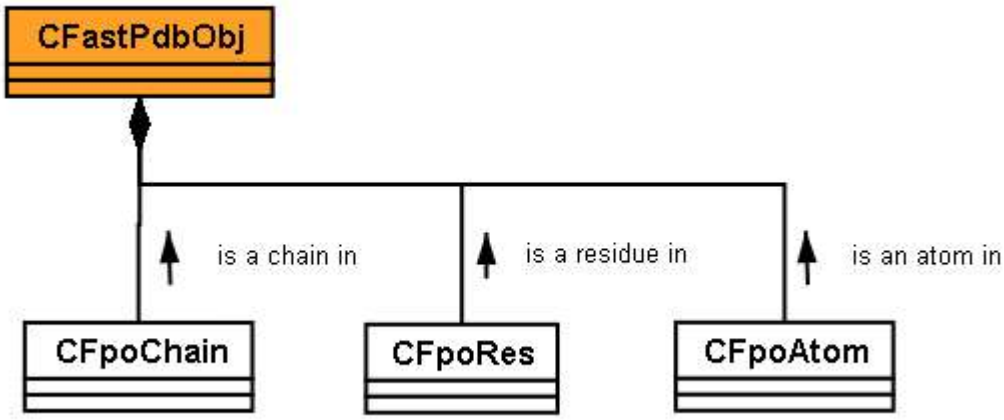

Fig. 11. UML diagram of CFastPdbObj and component classes.

12.2.3. Data Structure

PDB data files contain data that have a natural hierarchy. The lowest level data are ATOM records, which define the spatial locations of atoms, and some additional type information. Atoms are contained within residues, which are components of chains, which in turn are components within a single PDB model. Multiple PDB models may be contained within a single PDB file, as delimited by 'MODEL'/'ENDMDL' separators. Only the first model of a pdb file is read by CFastPdbObj.

Typical total atom counts for ribosomal subunit models are on the order of 50,000 atoms or more. A new CFpoAtom object is allocated for each parsed atom record. For performance purposes, pointers to all atom objects are stored in a single array in the pdb object. This was done to enable rapid searching of the atom set for certain features. An alternative would have been to store pointers to atoms within each residue, which would entail more hierarchical searches using nested 'for' loops.

Each residue object contains a count of the total number of atoms in the residue, and a numerical index to its starting atom, which is an offset in the single pdb-level atom list. The residue object also contains two arrays of bond information. The first is a list of intra-residue bonds, defined as pairs of canonical indices of atoms within the residue. The second list represents bonds to the previous residue in the chain, if any, and is defined as pairs of numerical offsets into the global pdb atom list.

Chain objects merely contain their PDB chain ID and index information about the residues that they contain. Consequently, the top-level pdb object has just three lists: a list of atoms, a list of residues, and a list of chains. In addition, the pdb object contains some limited information about the entire model, including the name of the file from which it was read, and optionally, TITLE record info and chain description info read from COMPND and MOL_ID records in the PDB file, if found.

12.2.4. Covalent Bond Creation

In the current implementation, CONECT records, which specify bonds between atoms, are not processed. One reason for this is that CONECT records are optional, and often not provided. Instead, after the atom data has been read, covalent bonds are created between

atoms using a simple algorithm that relies upon atomic valence and maximum bond lengths, as inferred from the atom's element type.

The algorithm works by iterating a set of atoms, and for each atom, it iterates the remaining atoms to test for possible covalent bond partners. However, this results in O (N^2) number of operations, which severely degrades performance for large atom sets. To avoid this performance penalty, the algorithm performs covalent bond creation in two separate phases. In the first phase, covalent bonds are created between atoms within a single residue. In the second phase, covalent bonds are created between residues.

While this approach works well for chains consisting of standard nucleic or amino acid residues, it will fail to create bonds between heterogeneous residues, such as a nucleic residue followed by an amino acid residue. It will also fail with non-standard residue types such as antibiotics. Since such non-standard connections are typically specified by explicit CONECT records, a future version of the algorithm will incorporate CONECT record information, if available, when creating covalent bonds.

One other limitation is that only single bonds between atoms are created. For static models, single bonds are sufficient for graphical presentation of structure. However, for dynamical modeling, more sophisticated processing is needed to define multiple bonds and to identify specific atom types and valence structure.

12.2.5. Data Access

The public class CFastPdbObj provides functions to access to the pdb data. Because this data represents a static model, the access is read-only, with a few, very limited exceptions. Functions exist to provide total numbers of chains, residues, atoms and

bonds. Additional functions provide chain ids, residue names and sequence numbers, atom positions, names and type information. This information is returned directly in value form, in response to general-purpose integer parameters that index a particular object, such as atom 8 of residue 22 of chain 5.

While this approach maintains good encapsulation and interoperability, it imposes a certain overhead and inflexibility that is unacceptable in certain situations, such as those requiring high-speed iteration of the entire set of atoms. An example of such a situation is the calculation of steric interactions between atoms, which is a common performance bottleneck.

Consequently, direct access to the internal list of atom pointers is also made available through certain functions in CFastPdbObj. However, when pointers are returned, they are returned using the CAtm or CPdbAtm interface in the wkChem library, so that direct exposure to the internal representation of CFpoAtom is avoided.

To simplify the complex internal representation of covalent bonds, a member BondIterator class is defined, which allows for traversal of all bonds in the object, returned by value via the CBnd interface.

12.3. wkEncon Library

The wkEncon library is designed to perform efficient calculations of energy and force resulting from pair-wise interactions of groups of atoms. Several classes in the library are used as components in the forcefield of the main program. Through the use of several interfaces, the library is able to work independently on the client data without being exposed to the implementation details and specialized classes of the forcefield.

The name of the library is derived from the initial intent for calculating the 'EN'ergy of various 'CON'formations of groups of atoms. Currently, the only type of energy and force calculations that are implemented are for steric (van der Waals) interactions. However, since the calculations for these types of interactions are generally $O(N^2)$, they can be a significant bottleneck for performance. In addition, it is necessary to consider the covalent bond structure, in order to skip interactions between 1-2, 1-3, and optionally 1-4 atom pairs. A number of classes and algorithms were developed for the library in order to address these considerations. Only the three classes that are currently used in the main program forcefield will be discussed. These are: CTorsionBodyTgaFcon, CTorsionBodyTgaxFcon and CTorsionBodyIteFcon.

12.3.1. CTorsionBodyTgaFcon

CTorsionBodyTgaFcon is a rather concatenated name intended to label the class in a way that distinguishes it from other classes in the library. The 'Fcon' portion of the name means that the class calculates the 'Forces' for a 'CONfiguration' of atoms, as opposed to an energy calculation. The 'Tga' part means that the configuration is divided into 'Two Groups of Atoms', in the sense that there are no covalent bonds between the groups. Finally, the term 'Torsion Body' means that the atoms in a group can rotate about their covalent bonds. This is in contrast to a 'Rigid Body', where all the atoms remain fixed, relative to each other.

For torsion bodies where every node is a single atom, an interaction between two atoms will produce a force, but no torque because the moment arms are zero. For a rigid body, a force applied at an atom will produce a torque on the center of mass of the entire body,

which is normally displaced from the atom center. To accommodate both rigid and torsion bodies, the wkEncon library defines an interface, CWkChemForceObject, which declares the function 'AddAtomForce()'. The wkEncon class passes the calculated force of an atom pair to the client through this function, allowing the client to accumulate the force and any resulting torque, as needed, either as a rigid or as a torsion body.

The interaction between two groups N and M is done by iterating all (An, Am) atom pairs and qualifying each pair with a threshold distance. The calculation of the steric force between atoms is done using the standard Lennard-Jones function as described in the section on forcefield implementation.

The class also provides options for calculating the total energy of the interaction, accumulating interaction counts, skipping hydrogen atoms, and outputting a detailed report of all interacting atom pairs. This final option is especially useful for generating steric interaction reports that show the significant atom overlaps and clashes in a conformation.

12.3.2. CTorsionBodyTgaxFcon

CTorsionBodyTgaxFcon is nearly identical to CTorsionBodyTgaFcon, but the additional 'x' in the name indicates an 'eXclusion' option, where a list of atom pairs should be excluded from interaction. This is typically used for calculating the steric forces between two adjacent residues in a chain, where atoms near the covalent bond joining the residues should be excluded because of their 1-2, 1-3 and 1-4 relationship.

12.3.3. CTorsionBodyIteFcon

To calculate steric interactions between atoms within a group, a more efficient approach is needed than a simple pair exclusion list. For a group of atoms of moderate size, such as the approximately 20 to 30 atoms within a nucleic acid residue, there will be a slightly larger number of covalent bonds, and the total number of exclusion pairs for 1-2, 1-3, and 1-4 relationships will be comparatively large. Such a list could still be implemented as a hash table for fast lookup, but it would still incur a performance penalty for the extra memory usage.

To address the need for efficient exclusion of atom pairs in intra-residue steric calculations, the CTorsionBodyIteFcon class was created. It uses an 'Iterative Token Exclusion' algorithm, in conjunction with a pre-existing list of bond definitions. The algorithm works to efficiently tag a subset of atoms for exclusion in each cycle of the outer loop of the pair iteration.

To iterate all atom pairs, each atom in an ordered sequence is considered for the first atom of a pair. This is the outer loop of the iteration. Then, the inner loop considers each subsequent atom in the sequence for the second atom of the pair, giving N * (N-1) iterations. At the start of each cycle in the outer loop, the algorithm recursively visits the bond partners of the first atom, to a desired depth L, and for each partner found, a flag is set in an array of all atoms. Then, during the inner loop, this array is used exclude interactions with the first atom at O(1) growth rate. The algorithm also has the advantage of including more closely bonded atoms, such as 1-4 atoms, by simply reducing the recursion depth parameter L.

12.4. wkTorsion Library

The wkTorsion library supports the process of creating and simulating dynamic models of molecules in the main application forcefield. The library provides a representation for atoms to undergo rotation about their covalent bonds. The development of the wkTorsion library was motivated by the Atom Group local frames method of (Zhang and Kavraki, 2002). That method was concerned with better ways to represent and analyze conformational change of molecules resulting from changes in torsion angles.

The Atom Group method is put forth as an improvement over two prior methods, the Simple Rotations method and the Denavit-Hartenberg (DH) local frames method. All of the methods begin by organizing the group of atoms as nodes in a tree representation, with some arbitrary atom as the root node. The covalent bonds are the edges that connect the nodes. The Simple Rotations method constantly updates all descendant atoms in a subtree with each rotation of a bond. A significant problem with this method is that small errors in rotation will permanently accumulate. The DH method avoids this problem by defining coordinate frames at the child end of each bond. Rotations are applied only to the frames, and the position of an atom is determined by combining the frames between it and the root node. However, the DH method uses the geometry at both ends of a covalent bond to define a local frame. The Atom Group method of Zhang et al. defines the frame at the child end in a more arbitrary way that reduces the number of frame combinations that are needed.

The wkTorsion library uses a representation that is similar to the Atom Groups method, except that the frames are initially defined as equal to the global reference frame, by

starting with an identity homogeneous transform matrix for each frame. For each rotation that is applied, the corresponding transformation is added to the frame matrix at one end of the rotating bond. Also, the allocated memory for the transform frames is not provided within the library itself, but instead is supplied externally by a client. This gives greater flexibility and lower memory overhead. The library consists of just three classes: CTorGraph, CTorNode and CTorAtom, as show in Figure 12.

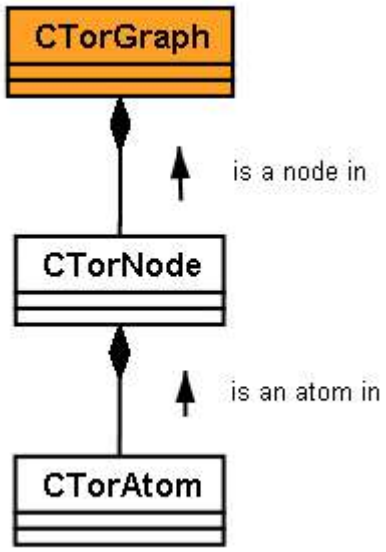

Fig. 12. UML diagram of relationship
between classes in wkTorsion library.

12.4.1. CTorGraph

The CTorGraph class is used to create a tree representation of atoms from an input set of atoms and bonds. Nodes are created by recursively following covalent bonds from a root atom. An option exists to consolidate ring structures into a single node, although recent implementations of the forcefield have been more successful by creating just a single atom per node. The class provides utility functions for iterating nodes, applying rotations

at either side of a covalent bond, updating atom positions, and periodically normalizing the rotation matrices of the node frames, which is important for correcting accumulation of errors in rotation.

12.4.2. CTorNode

The CTorNode class was created in order to make a clear distinction between a node in the torsion graph and the one or more atoms that are located in that node. This distinction is more significant in the case where there are multiple atoms consolidated into a single node, as can be done for a ring structure. However, even in the case of a single atom node, the attributes involved in graph operations, such as ancestor/descendant relationships, are more applicable to nodes than atoms.

12.4.3. CTorAtom

A CTorAtom is an object that wraps an 'unmoved' atom. The unmoved atom is typically from a static model and provides a 'constant' position. Only the torsion atom position is changed by rotations applied to the torsion graph. The torsion atom position results from the transformation of the 'unmoved' atom by the combined sequence of frames between the torsion atom node and the root atom node.

12.4.4. Creating Torsion Graphs for Dynamic Models

When creating torsion graphs for dynamic models in the main application, best results were found when the total number of atoms within a single torsion graph were fairly small. The first use of a torsion graph was to create a single graph for all the atoms in a PDB model. While successful, it was noted that the creation time was very slow, due to the large amount of recursion during node creation and consolidation. Node traversal also

resulted in a slowdown for drawing and manual rotations of the graph. Later, it was found that creating torsion graphs at the residue level provided far better performance. The relatively small number of atoms in the graph (corresponding to around 20 - 40 atoms in a residue) significantly reduced the number of node traversal operations.

12.5. wkGrMatrix Library

The wkGrMatrix library defines classes for vector, matrix and homogeneous transform operations. The initial purpose of the library was to support 3D graphics applications, which explains the 'Gr' portion of the library name. Over time, the library has also been used for some non-graphical, purely geometrical tasks. The library consists of three principal classes: CGrVector, CGrMatrix and CGrTransform, as shown in Figure 13.

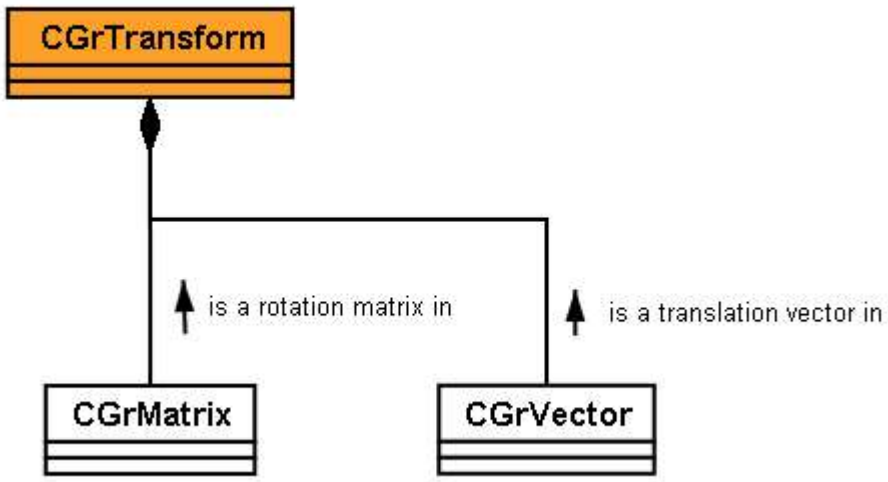

Fig. 13. UML diagram of CGrTransform and component classes.

12.5.1. CGrVector

For simulation of physical models, a three-dimensional coordinate vector is a widely used and fundamental element for computations involving position, velocity, force and many other physical attributes. The representation of this data, along with numerous vector-related operations, is defined in the class CGrVector.

For vector operations, there are many well-designed open source class libraries that are freely available for reuse. The csVector3 class, from the Crystal Space project (Linux Magazine, 2003), was used as the original source for the CGrVector3f class. The class was extended to support both floating point and double precision. Following the naming practice of the OpenGL library, the precision of a particular class is indicated with either a '3f' or '3d' extension.

12.5.2. CGrMatrix

The CGrMatrix class defines a 3x3 matrix that represents an orientation in 3D space. The class serves as a component for the CGrTransform class, discussed below. By maintaining the rotation matrix as a distinct component, certain operations can be done more efficiently than in the composite format of the 4x4 homogeneous transform matrix. These operations include computing the matrix inverse and concatenating rotations.

12.5.3. CGrTransform

The CGrTransform class is used to perform a transformation operation, which maps a point in 3D space into another location in that space (Angel, 1997). The class defines a 4x4 homogeneous transform matrix, which has a rotation component and a translation

component. When combined in the matrix, they represent a coordinate frame that can be transformed to arbitrary positions and orientations in space.

The CGrTransform class is commonly used to instantiate an object that represents the location of a static model of atoms. A succession of translations and rotations can be accumulated in the transform object instead of being applied directly to the model. The actual transformation of all the atoms in the model can be done as a separate operation, as needed. This has numerous advantages for static modeling, in which the relative locations of the atoms in the model do not change. One example is when the user inserts and moves multiple large PDB models in a document. To save the adjusted configuration, it is only necessary to save the transform matrices of the models, along with a reference to the original PDB file names. The alternative would be to write out a new PDB file with the adjusted atom locations, which may require the saving of hundreds of thousands of ATOM records. The CGrTransform is also used to represent the node frames in torsion graphs in the wkTorsion library.

12.6. wkGraphic Library

The wkGraphic library defines some basic interfaces for drawing operations, which proved very important for isolating graphical from non-graphical areas of the project architecture, as discussed earlier. The library also contains some miscellaneous graphical services that are useful to the main application.

12.6.1. OpenGL

The low-level graphical output of the project, including the wkGraphic library and other graphics modules, is done using OpenGL, which is a software interface to graphics hardware (Woo et al., 1999). The library is designed to support interactive 3D graphics. It is relatively low-level, in that the user works with geometric primitives such as points, lines and polygons. OpenGL is portable across many operating systems and is widely used for scientific software. There are other graphical interface libraries available, such as DirectX, a Microsoft product. However, DirectX is currently confined to Microsoft Windows platforms and is mostly used for computer games.

12.6.2. CWkGraphicObject

CWkGraphicObject is an interface for objects to produce graphical output. At its core, the interface defines a Draw() function that is typically called during the graphical update event of the main application. Objects that implement the interface will perform any drawing required of them inside of this Draw() function. Because this class is strictly an interface, no member data is defined and there is no memory overhead incurred by inheriting from it.

An important function of the interface is to reduce linkage between graphical modules of the project. It attempts to do this by defining the some of the more basic elements that would be common to these modules. However, what can be considered basic will probably always be a matter of opinion and domain interest. In the limited context of this project, the basic elements that were retained, after several phases of evolution of the

project architecture, were functions to access a name associated with a graphics object, visibility flags, wireframe and transparency rendering, and the use of a selection color for displaying the selected state of objects in the graphics window.

12.6.3. CWkGraphicContext

An associated interface, CWkGraphicContext, is implemented by the caller to define a global graphical state. A reference to the graphics context is provided as a parameter in the Draw() function of the CWkGraphicObject interface. Specialization of the graphics context is left to the particular application, but a number of basic attributes and operations are pre-defined. These include functions to increment the count of drawn polygons, which is useful for tracking graphical performance. Viewport state is another attribute that can be made available. The size and direction of the viewport are used in frustum culling, which is a graphics optimization technique that avoids unnecessary drawing.

12.6.4. CWkSmoothColorMesh

An important feature in the main application of the project is the graphical display of molecular surfaces. In conjunction with other graphical representations of molecular structure, display of the surface is very useful for indicating steric boundaries, docking interactions, and visual cues of structural motifs.

Generation and display of the graphical primitives that represent a molecular surface can be quite computationally intensive, depending upon the size of the model and resolution of detail. The algorithms and process for generating the data needed to display a molecular surface are implemented in the wkGrChem library, discussed in more detail below. The result of that process is a called a 'polygon mesh', which is a list of vertices

and a set of outlines (Moller and Haines, 1999). The mesh is used to draw a set of connected triangles that form a closed surface.

The CWkSmoothColorMesh class is used to efficiently draw a mesh object. Efficiency is important because a surface for a model of the large ribosome subunit, drawn at a resolution of 2 Angstroms, will result in a mesh of 310,000 triangles as shown in Figure 14. Although modern graphics cards now have hardware-accelerated transform and lighting support, it is still necessary to present the data efficiently to the graphics pipeline.

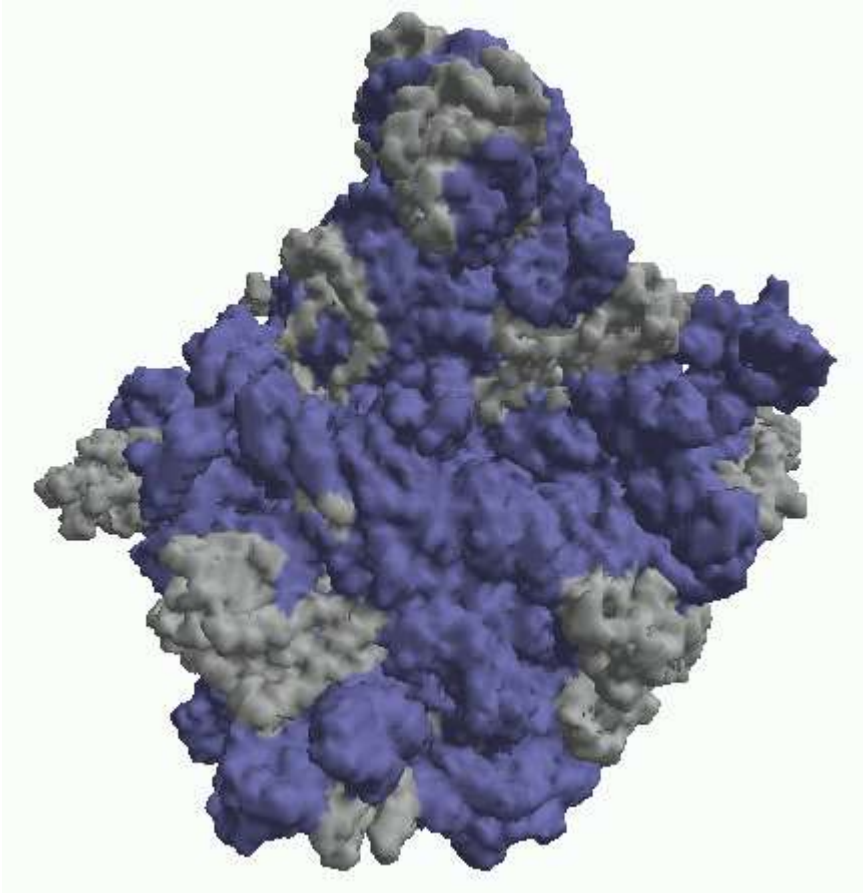

Fig. 14. CWkSmoothColorMesh rendering of 2 Angstrom surface mesh of PDB model 1FFK.

The set of triangles that enclose the molecular surface could be drawn with a flat surface for each triangle. This would result in a surface composed of numerous small facets and hard edges. However, OpenGL allows drawing operations to submit averaged surface normal vectors, so that the surface will have a 'smoothed' appearance, which is a more natural representation of electron density and solvent accessibility. In addition to normal smoothing, the color defined at each vertex is averaged with colors at neighboring vertices. Color smoothing improves the highlighting of various regions of a molecule so that they stand out visually from the surroundings.

Molecular surfaces can convey a lot of visual detail, but if they are opaque they will completely obscure any internal structure. Effectively communicating the details of a complex 3D structure is a significant challenge for molecular graphics software. Displaying all or part of a molecular surface with a variable degree of transparency is helpful for simultaneously conveying the steric boundaries and the internal skeleton of a structure.

The CWkSmoothColorMesh class implements the transparent rendering of its component surface, using a scalar value from a client that specifies the degree of transparency, ranging from 0 to 1. The option can be applied globally, so that every triangle in the surface is transparent, or it can be color-keyed, so that only those triangles with a specified color are drawn transparently. This has the effect of selectively opening a 'window' in an otherwise opaque surface.

The current implementation of transparent drawing suffers from a limitation. In order for the transparency to be perceived, the OpenGL depth buffer must be disabled. The depth buffer is a mechanism that insures that pixels closer to the viewer will properly obscure

pixels that are farther away. Since the depth buffer must enabled for normal rendering, all transparent drawing in the application is done in a separate drawing pass during the graphic update. Each object does its own drawing in this separate pass, independently of other objects. To effect proper occlusion of far pixels by near pixels when the depth buffer is disabled, the 'painter's algorithm' (Savchenko, 2000) is used. This algorithm works by drawing objects in a depth-sorted order, so that far triangles are drawn first, and closer triangles are drawn subsequently.

The problem in this case is that the depth-sorting is only done on a per-object basis. If multiple transparent surfaces are displayed, their depth appearance is not correct. The problem can be solved by implementing a more global sorting of transparent primitives from all objects, but this solution has not yet been applied to the current graphics architecture. An additional calculation is required when sorting the triangles. Because they must be sorted with respect to the current direction of the viewer, the existing algorithm employs an approximation that chooses the global coordinate axis that is most closely aligned with the current viewing direction and sorts the triangles in the appropriate direction on that global axis.

12.7. wkGrChem Library

The wkGrChem library produces graphics for chemical objects. This makes it a rather specialized library, but it fills an important role of allowing a clean separation of graphical operations from the main body of code for processing chemical data, at least at the library level.

The library has classes for drawing both static and dynamic molecular models. It can produce simultaneous representations for a number of different aspects of molecular structure, including space-filling spheres, covalent bonds, chain backbones and molecular surfaces. In addition to displaying images of chemical objects, the library supports the OpenGL mechanism for picking objects from mouse clicks on the rendered image. The primary emphasis of the library is to produce real-time, interactive 3D graphics, but there is also support for exporting a graphics scene to external formats such as POVRAY and VRML.

The principal classes in the library that will be discussed include CGrAtmUtil, CGrPdbObj, CGrPdbSurfFactory, and CGrTorGraph.

12.7.1. CGrAtmUtil

CGrAtmUtil is a utility class that draws spacefill atoms and covalent bonds. It can draw individual atoms and bonds, or groups of them. When drawing groups, the class will can exploit several opportunities for optimizing the drawing operations. Another requirement that it supports is drawing only one side of a covalent bond. This is necessary for situations where the atom on one side of a bond is visible, but the atom on the other side is not. The radius of the bonds and spheres is specifiable by the client. The color used can either be a user-defined color, or a default CPK (Corey-Pauling-Kultun) color that is associated with the element type of the atom.

When drawing atom spheres, the class uses the OpenGL utility function gluSphere(). Although this tends to be optimized in current graphics cards, the number of spheres, and consequently the total number of polygons needed, can become a performance issue. For

example, drawing all spacefill atoms at the default spherical quality for the small ribosomal subunit of PDB model 1j5e, as shown in Figure 15, will result in a polygon count of 3.3 million.

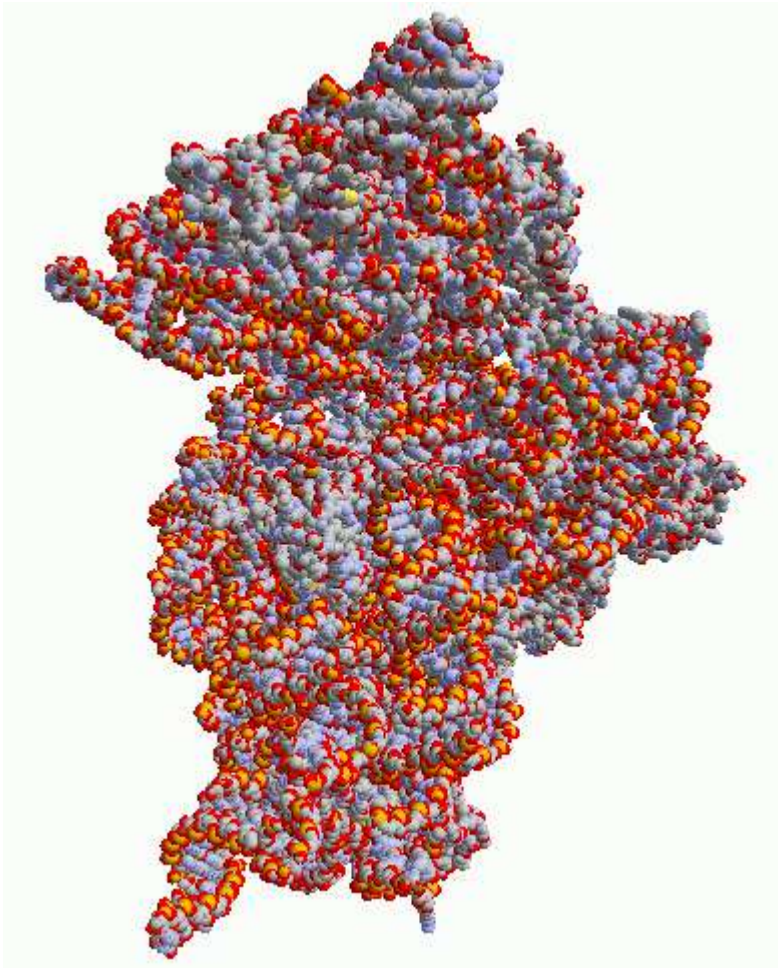

Fig. 15. Spacefill rendering of small ribosomal subunit by CGrAtmUtil.

12.7.2. CGrPdbObj

The CGrPdbObj class is used to create graphical representations of a static PDB model. A CGrPdbObj object receives and retains a reference to a CFastPdbObj object, which is the

source of the model data. During a drawing event, a CGrPdbObj object will obtain the positional and chemical data from its associated CFastPdbObj object, and then use the data to display the structure of the model. The relationships between the CGrPdbObj class and its associated classes is shown in Figure 16.

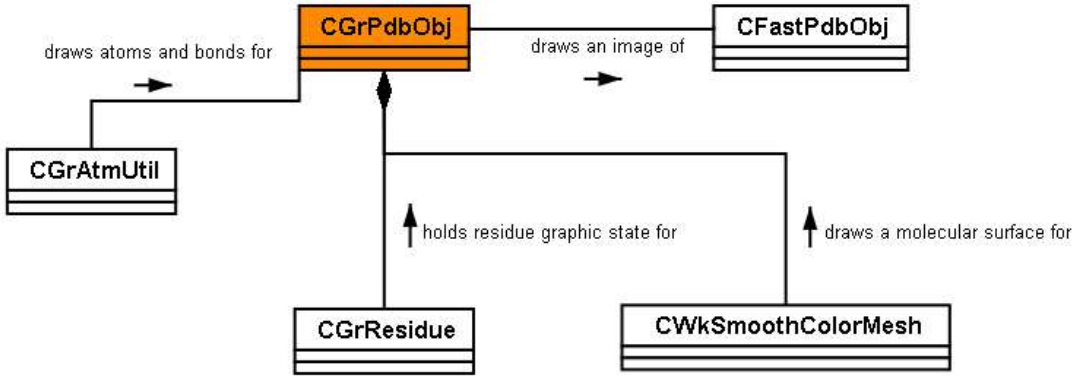

Fig. 16. UML diagram for CGrPdbObj class and related classes.

In addition to displaying various representations of the model, such as covalent bonds, chain backbones and molecular surfaces, the CGrPdbObj class manages a 'selected' state for the components of the model. This allows a user interacting with the graphical representation of the model to select and deselect its component chains and residues. The selected state can be indicated by displaying the selected subset in a different color from the rest of the model.

12.7.3. CGrPdbSurfFactory

A CWkSmoothColorMesh object is used to draw the molecular surfaces for a PDB model. CWkSmoothColorMesh is a general-purpose graphics class in the wkGraphic

library, discussed above. However, its surface mesh data is created by a specialized class in the wkGrChem library, CGrPdbSurfFactory.

12.7.4. Marching Cubes Algorithm

CGrPdbSurfFactory uses an implementation of the Marching Cubes algorithm, a method for efficiently constructing a three dimensional surface around an arbitrary structural model (Lorensen and Cline, 1987). The algorithm was originally used for producing images from 3D medical data such as NMR scans.

The algorithm works by subdividing the volume containing a model into smaller cubical subvolumes, where the size of the cubes is determined by the desired resolution of the surface. Smaller cubes will produce a more detailed surface, but will increase the computational time and memory usage. Every cube within the volume is then examined for intersection with the target model. Intersection with the model is tested by examining the locations of the 8 vertices of the cube. Each vertex is a assigned a binary value of 1 or 0, depending on whether it is inside or outside of the model. This results in $2^8 = 256$ possible states for a cube.

The next step in the algorithm is to define triangles within a cube in order to enclose the vertices that intersect the model. The vertices of the triangles are simply determined by locating them at the midpoints of the cube edges. Figure 17 shows the triangles that are constructed for a few of the possible cases.

Because of symmetry, it is possible to reproduce the 256 cases by defining a much smaller number of cases and applying rotation and inversion operations. However, in this

project, all 256 cases were manually defined, with the assistance of a 'case viewer' script that graphically represented them. Although a bit tedious, this approach was straightforward in implementation and efficient in operation.

There is one deficiency in the original Marching Cubes algorithm. It is referred to as the 'hole ambiguity' problem. It arises out of a particular configuration of two adjacent cubes which have a complementary set of polygons in such a way as to produce a 'hole' in the

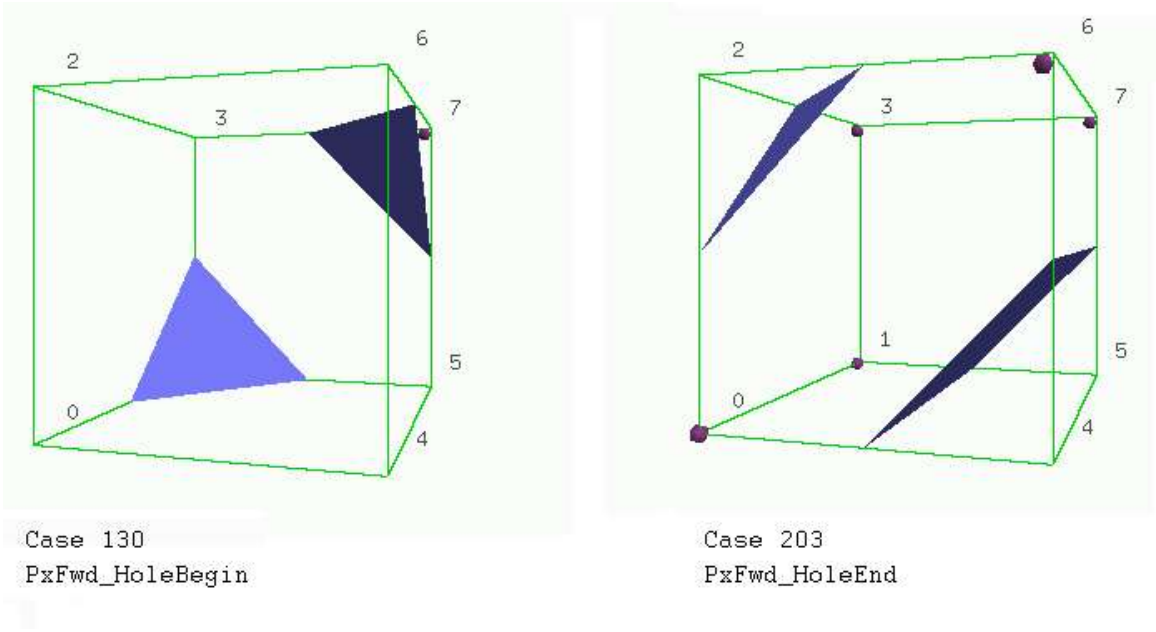

Fig. 17. Two marching cube cases. Each case defines polygons that are needed to cover occupied cube vertices.

surface as shown in Figure 18. The number of instances in which this can occur is just a small fraction of the total number of possible configurations. Although the visual artifacts that result are fairly small in comparison to the total surface, they can be quite aesthetically disturbing and so a modification was made to the algorithm to 'patch' the holes.

The hole configuration occurs when one side of a cube has two diagonal vertices in the 'filled' state and the other two diagonal vertices in the 'empty' state. Furthermore, a particular configuration of polygons is required such that two parallel edges are defined on the cube face, and the other two edges are defined on the face by the adjacent cube. The two cubes cases shown in Figure 17, if stacked against each other, are an example of a hole configuration. The particular stacking alignment is defined when vertices (1, 7) of case 130 are coincident with vertices (0, 6) of case 203, respectively.

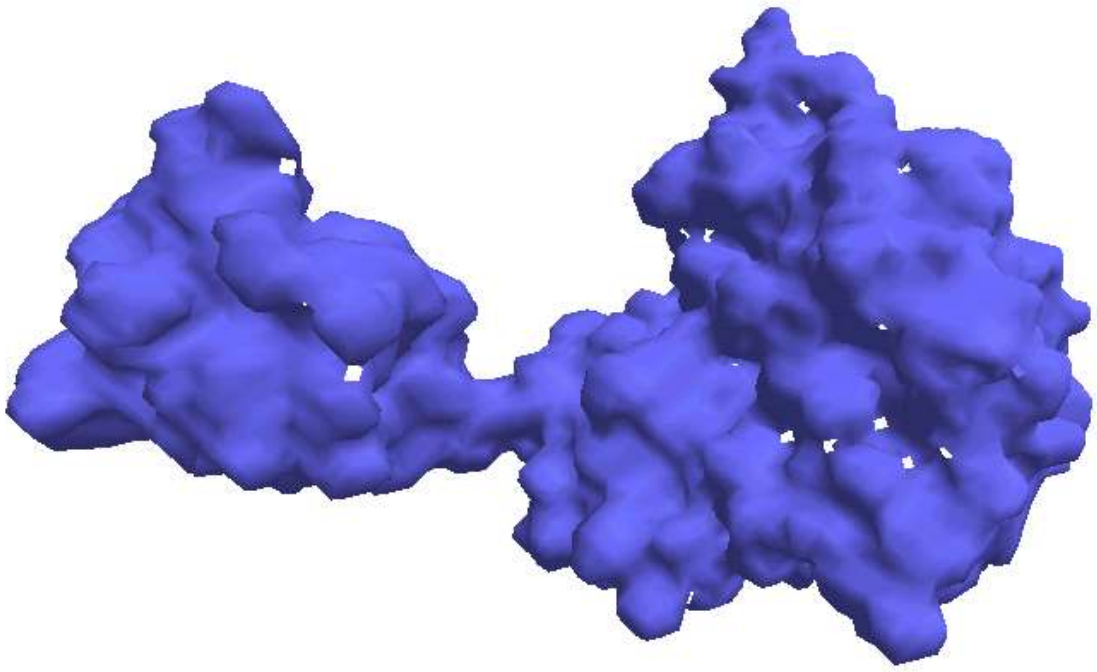

Fig. 18. Hole artifacts in surface generated by the marching cubes algorithm.

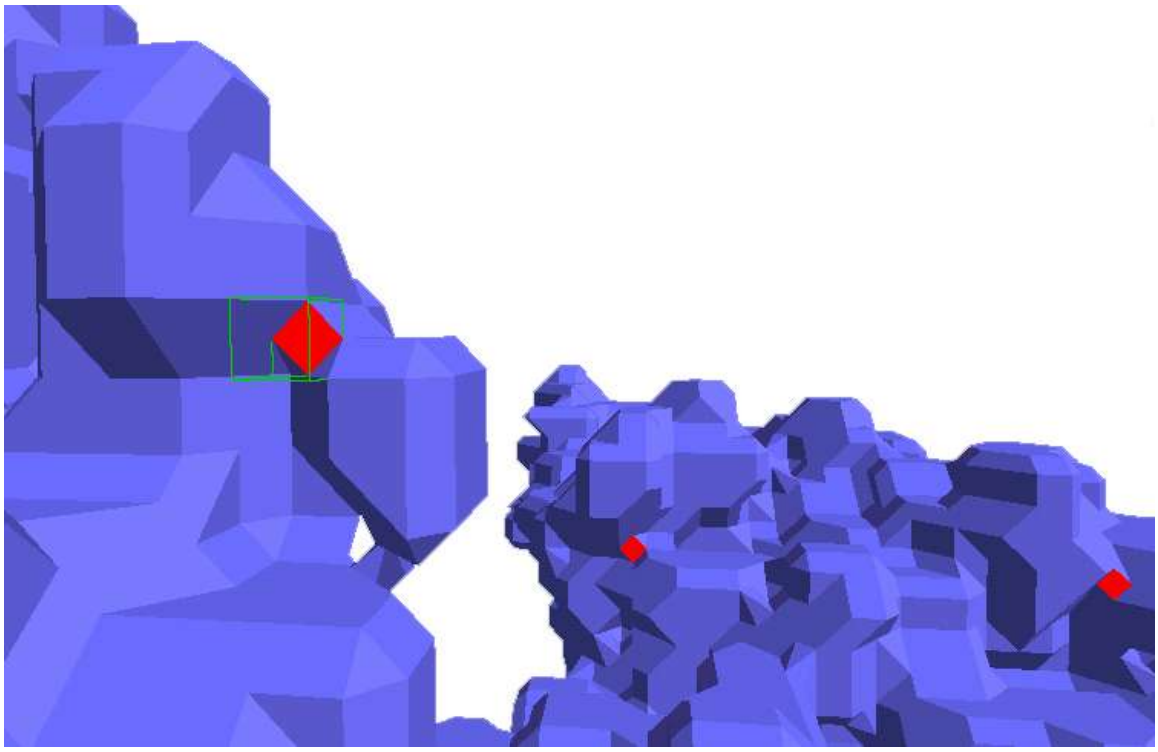

Fig. 19. Patched version of marching cubes algorithm. The hole patching polygons are highlighted in red. Surface smoothing has been turned off in order to show the polygon boundaries.

The patch algorithm works by looking for a match of the limited number of cases of two adjacent cubes that will produce a hole, and if found, defines an extra polygon in the face to fill the hole as shown in Figure 19.

12.7.5. CGrTorGraph

Similar in purpose to CGrPdbObj, CGrTorGraph is a class that produces graphic representations of molecular structures. The difference is that CGrTorGraph is used for drawing dynamic models. The UML diagram in Figure 20 shows its primary structural relationships. The source of model data comes from CTorGraph, which represents a flexible structure of bonded atoms.

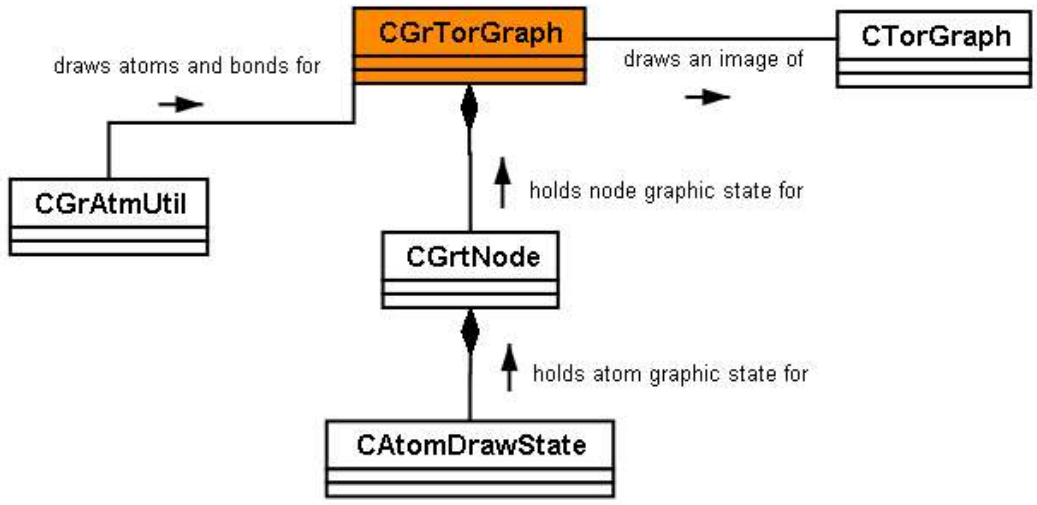

Fig. 20. UML diagram of CGrTorGraph and related classes

In contrast to the current implementation of static model drawing, which only allows specification of graphic state at the residue level, CGrTorGraph can draw and maintain selection state for individual atoms. A limitation of CGrTorGraph is that it does not provide a surface representation for dynamic models. The constantly changing geometry of such models would require regeneration of the surface mesh data for each timestep, which would be too computationally expensive in the current implementation.

12.7.6. Graphics Export

On occasion, modeling work done in the project needed to be exported to high-quality graphics suitable for publication. This was done by creating a group of classes that could translate the internal graphical representation into a common format supported by other software tools. A graphics export interface class, CExportGraphics, was defined along with two derived classes, CVrmlExportGraphics and CPovRayExportGraphics.

12.7.6.1. VRML

VRML (Virtual Reality Modeling Language) is a specification that, among other things, defines an interchange format for 3D geometrical modeling features (Carey and Bell, 1997). Because it is an ISO standard, the VRML file format is used as a way of exchanging 3D modeling data between different software tools. Although a newer specification, X3D, has recently been standardized, it retains VRML at its core (Parisi, 2004).

The CVrmlExportGraphics class was created to produce VRML data for a subset of graphical representations in the project. After some limited use in the early stages of graphical export, it has currently been superceded by direct export to POVRAY, discussed below. However, support for VRML in the project is important for more than just publication of 2D graphics. Its capabilities for specifying object behaviors make it suitable for applications in schematic simulation (discussed below) and macromolecular movements (Krebs and Gerstein, 2000). For these reasons, much greater support for VRML is expected in subsequent project development.

12.7.6.2. POV-Ray

POV-Ray is a highly developed and freely available software tool for creating high quality 3D graphic images (Povray Org, 2005). It uses the rendering technique of ray-tracing, which reproduces the effects of light in a more realistic way than methods designed for interactive graphics, such as those used in OpenGL.

POV-Ray export classes have been defined to generate POV-Ray code to reproduce the currently displayed graphics in the application main window. The classes receive references to model data from the main application during an export operation. This data allows the POV-Ray classes to obtain the graphical and geometrical state information and translate it into a sequence of text commands that are output to a .pov file. The file can then be loaded into the POV-Ray rendering program and used to produce a high quality image. Figure 21 shows side-by-side images of the anti-codon stem loop of a tRNA molecule. The image on the left is from a screen capture of the graphics window of the application. The image on the right is the POV-Ray version.

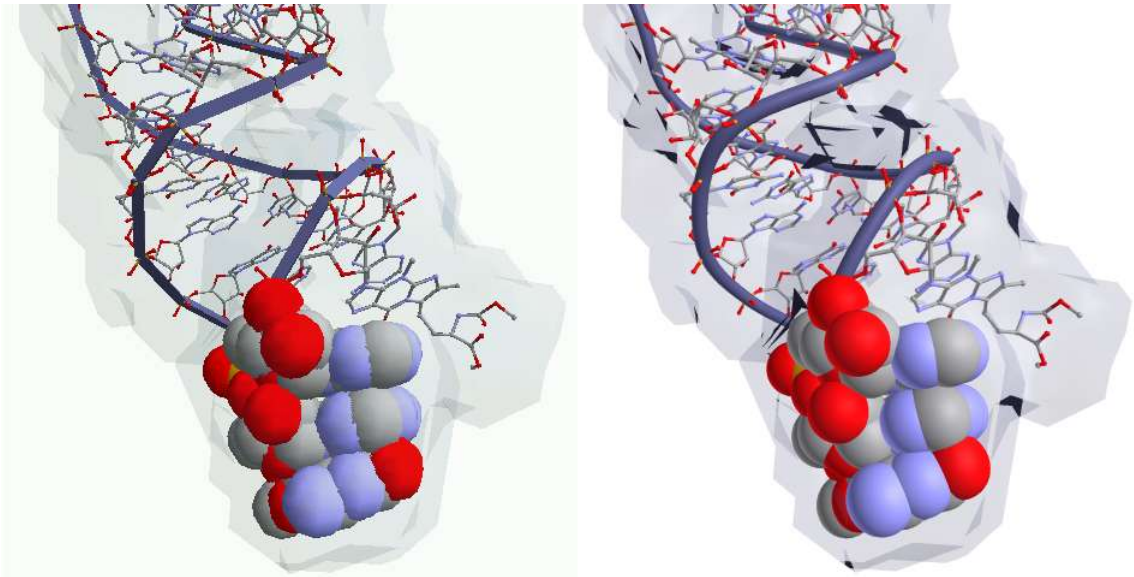

Fig. 21. Comparison of rendering of tRNA anticodon-stem-loop. Left side: rendered with Ribosome Builder OpenGL graphics engine. Right side: rendered with POV-Ray program, version 3.5.

Note the smoother and better looking results of the POV-Ray version, in most respects. One exception is in the area of multiple overlaps of the surface mesh, which are too dark in the POV-Ray image. Currently, only static models can be exported, but support for dynamic models should be fairly simple to implement.

12.8. wkGeom Library

The wkGeom library contains classes for geometrical operations. The library currently consists of just three classes: CExtent, CQHull and CWkGeomUtil. CExtent defines an axially-aligned bounding box, an important construct for detecting the intersection of three dimensional objects. CQHull is a wrapper for functions in the third-party quick hull library (Barber et al., 1996). This library was used in early versions of the program to generate convex polygonal surfaces around PDB residues, but is not currently used. However, the qhull library remains part of the current build and distribution for possible future use. CWkGeomUtil contains miscellaneous functions for intersection testing, curves and interpolation.

12.9. FPSocket Library

The FPSocket library contains low-level classes that are used in creating an interface between the main application and an external web browser. These two applications are processes that are both running on the same local host computer. Communication between them can take place through the creation of a network socket connection. The flow of data over the connection is determined by a socket interface, which is an API (Application Programming Interface) for TCP/IP networks (Jamsa and Cope, 1995). The FPSocket library contains the class FPSock_SimHttpSrv, which functions as a simple HTTP server that will respond to GET requests from an HTTP web browser. This server class forms the basis for a derived class in the htsrv application, a standalone process that receives web browsers requests and transmits them to the main Ribosome

Builder application. The operation of the htsrv application is discussed in more detail below.

The initial version of the FPSocket library was created by Drew Kearns, and obtained via personal communication. This third-party version only contained classes for client-side communication. The listener and HTTP server classes were subsequently added in development of the Ribosome Builder project.

12.10. wkParse Library

The wkParse library provides some low-level functions that assist in the interpretation and execution of commands in the main application. The definition and processing of commands is an important part of the program. Command processing is the fundamental mechanism for the user to control the application in order to achieve desired results.

The multiple alternatives for inputting commands, such as menus, toolbars, hotkeys and console input all rely on a common set of command definitions. The commands take the form of text strings that consist of command names along with a possible sequence of parameters. At present, there are more than 250 distinct commands that have been defined. The wkParse library is used to implement a particular strategy for effectively processing these commands for the main application.

Early versions of the program defined commands in a very ad hoc manner. Whenever a new feature was added, a command needed to control the feature would be implemented by adding blocks of code to the existing module for processing commands. The problem was that the new code was added in a variety of inconsistent ways and places, sprinkled throughout the module in the form of text strings, if statements, case blocks and

functions. There was a great deal of duplicated code associated with extracting and validating command parameters. Documentation for new commands was located in a separate location, and sometimes failed to be kept in sync. The result was a command processing scheme that was inefficient, difficult to maintain, and inconsistently documented.

There are well developed techniques in computer science for defining and interpreting command strings. Language grammars, lexical analysis and parsers can be used to precisely extract, analyze and translate complex strings of tokens in order to implement a command language (Aho et al., 1988). When a new version of command processing was being developed for the Ribosome Builder project, the use of this more rigorous approach was considered. However, a number of factors such as lack of time, formal training and experience necessitated the choice of a simpler, yet effective method. In addition, the subsequent embedding of a pre-existing scripting language and interpreter tended to compensate for the deficiencies of this simpler method.

The new method for command processing relies upon a hierarchical organization of commands, in which each command name may serve as a container for a subset of related commands. This approach is similar to the hierarchical organization of menus and menu items in graphical user interfaces. For example, a top level command in the hierarchy is the 'draw' command group, which contains commands related to the graphical display of objects. A subcommand of this group is the 'covalent' command, which can be used to adjust the display of covalent bonds. The command string 'draw covalent on' uses two command names and a parameter to specify that covalent bonds should be shown. In addition to terminal commands such as 'covalent', the 'draw' command contains

subcommands that are themselves command groups. An example is 'draw surf', which contains subcommands for drawing molecular surfaces.

A second aspect of the method involves definition of command syntax. The syntax for each command is defined by a set of fields that declare the command name, the number and type of allowed parameters, a usage string and a longer description string. These last two fields enable the automatic generation of a minimal amount of documentation for every command in the application. The set of fields for each command are contained within a C-type structure that is declared as static data in the main application. A number of structure elements are defined sequentially as array items which define the command group. These are subsequently associated with the parent command for that group. Each parent command is associated with a member function in the command processor class of the application.

The CCmdParser class in the wkParse library will receive a command from the main application, in conjunction with the set of command definitions. It will then attempt to parse and dispatch that command to the appropriate member function of the application command processor. In addition, the CCmdParser will extract and validate any arguments in the command. These arguments are made available to the application member function through a CCmdLine object, which is passed as a parameter. The wkParse library classes CCmdParser, CCmdProcessor, CCmdLine, CmdDef and ArgDef are shown in Figure 22, along with their relationship to the client application.

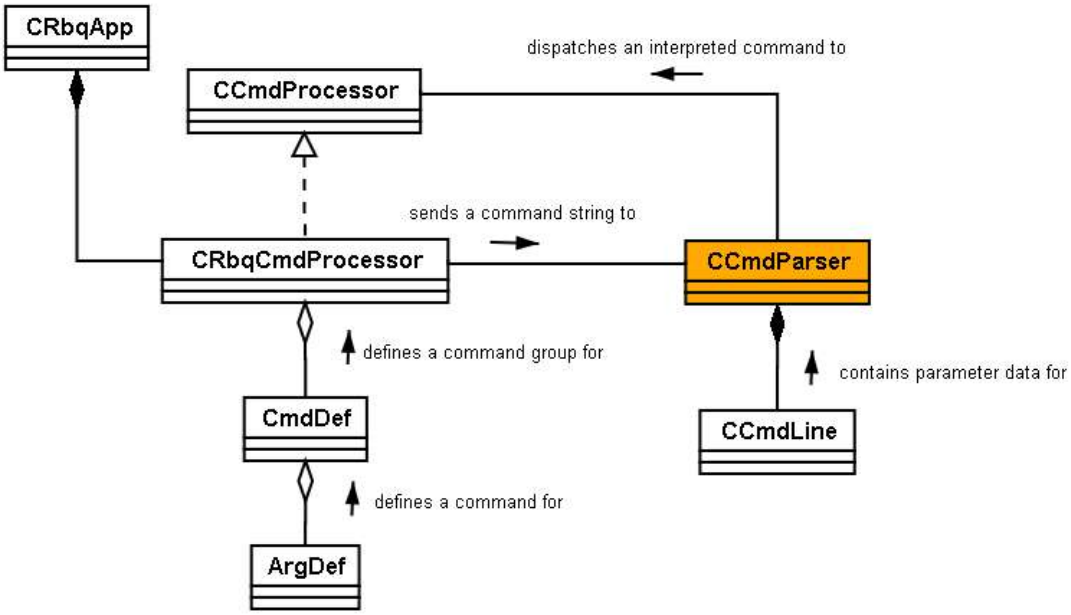

Fig. 22. UML diagram of CCmdParser and related classes.

12.11. wklib Library

The final low-level library, wklib, is the oldest and most general purpose. It contains groups of functions that are used for debugging, file, string, math, vector and OpenGL operations. The vector functions were used before the adoption of the object-oriented version in CGrMatrix, and are still found in certain areas of the project.

Chapter 13

Application-Specific Libraries

The low-level libraries do not have any dependencies on the main application, and in consequence they can be re-used in another application. There are three additional libraries that have a closer connection to the Ribosome Builder application. As a result, the attempt to re-use of one of them is more likely to drag in the others because of include file dependencies. These three application-specific libraries are rbqUtil, rbqLua and frameWk.

13.1. rbqUtil Library

In addition to some application-specific utility functions, the rbqUtil library defines the CRbqGraphicObject class, which is an implementation of the CWkGraphicObject interface. A number of specialized graphics classes are also defined to display spheres, boxes, lines and other building blocks that are used for creating annotations in the main application. CRbqGraphicContainer is defined as a container class for these graphical components, and helps to manage access to them in the graphics object hierarchy.

13.2. rbqLua Library

The rbqLua library embeds an extension language, Lua, into the main program. This allows the user to write scripts in the extension language and submit them to the program for execution. The scripts are able to make use of an interface, which consists of a large

number of functions that provide access to many parts of the main application. The rbqLua library currently consists of two classes, CRbqLua and CRbqScriptObject.

13.2.1. Extension Languages

In the last decade especially, a number of the commonly used computer programming languages can be classified into two distinct groups. These groups are referred to as 'systems programming languages' and 'scripting languages' by Ousterhout (Ousterhout, 1998). Ierusalimschy (Ierusalimschy et al., 1996) refer to them as 'general-purpose programming languages' and 'extension languages'.

Regardless of the labels, the characteristics that distinguish them are fairly clear. System programming languages such as Fortran, C, C++ and Java are designed for working with fundamental data structures and algorithms. In addition, they are strongly typed to help manage complexity. In contrast, scripting languages such as Perl, Tcl, Python and Lua are designed for gluing fundamental components together. They are weakly typed, to enable rapid prototyping and application development.

At the risk of some imprecision, an analogy may be helpful to those who are not as familiar with the meanings and implications of some of these computer science concepts. Working with a systems programming language may be compared to the technologies used to construct large permanent buildings. This would include the equipment and procedures to do surveying, grading and leveling, laying concrete foundations, and installation of carefully designed components for plumbing, electricity, elevators and many other subsystems. In order for such complex projects to be successful, a great deal of planning and adherence to standards and specifications is necessary.

A scripting language, on the other hand, is more like the technology used to establish a campground, concert, carnival or flea market. Lightweight tents can be quickly setup on existing terrain. Portable toilets, electrical generators and extension cords can be rapidly deployed. Little pre-planning is required, and it is much easier to make changes and adaptations. Although there is great speed and flexibility with this approach, the resulting 'tent cities' are not as efficient, and they are more vulnerable to storms, electrical fires and sanitation problems than are the more slowly-built permanent structures.

13.2.2. Lua

Lua is a small, robust and extensible cross-platform open source extension language with powerful data definition and object-orientation capabilities. In addition, its syntax is simple and elegant and intended to appeal to the non-professional programmer. The source code is freely available and can be obtained from http://www.lua.org. Lua version 4.01 is the version used in the Ribosome Builder program. A concise but comprehensive reference manual for this version is located at http://www.lua.org/manual/4.0/manual.html.

13.2.3. CRbqLua

The source code needed to compile and link the Lua interpreter and standard libraries into the main application is included in the project distribution and located with the low-level libraries. The rbqLua library defines a wrapper class, CRbqLua, that manages access to the Lua environment. Within the module for this class, there are also around 200 static function definitions written in C that are registered with the Lua environment. These

functions comprise the interface between Lua scripts and the main application. They are discussed in more detail in the Scripting API section below.

Where possible, the script API functions will access the main application by way of the application processor, CRbqCmdProcessor. This is desirable because it maintains a 'single point of access' that promotes consistency and lessens redundancy. However, because the script interface is more programmable than the interactive user interface, there are situations in which the script API must make direct access to some other part of the main application code, because no corresponding internal command has yet been defined for that case.

13.2.4. CRbqScriptObject

A second class in the rbqLua library is the CRbqScriptObject class. This class was created to address the need for a persistent object in the scripting environment. Script objects are discussed in more detail below, but briefly, they can distinguished from scripts that are intended to run to completion in a single instance of execution. The CRbqScriptObject class provides a means for defining a script object that can persist until it is explicitly destroyed. In addition, it is possible to connect the script object to a number of events that occur in the application, such as drawing, keyboard input and periodic timer ticks. However, the current implementation of script objects, and especially the means for defining and accessing script object variables, is rather cumbersome. Some recent alternatives for persistence in the Lua environment are in the process of being developed. One example is the AnnUtil.Sim object, discussed in more detail in the section on schematic simulations.

13.3. frameWk Library

The frameWk library contains classes for managing transform and selection operations in the 3D graphics window. Specifically, it provides a mechanism for translating mouse movements from the user into rotations and translations of selected objects and the view. In addition, mouse clicks on images in the window are used to select the objects represented by those images.

The name frameWk was chosen because the library makes good use of interfaces which allow the frameWk classes to perform these services for a client application without having any direct linkage to the specific implementation of the client classes. This enables good reuse of these mechanisms in other 3D applications that require transformation and selection operations in a 3D space. The name 'frameWk' is actually a little too generic, but the more descriptive alternative, 'threeDimensionalTransformAndSelectionFramework' is too long. At present, the library only uses input from a mouse. 3D input devices, such as SpaceBalls and data gloves, would be supported by the addition of similar classes to this library.

13.3.1. Moving Objects

The ability of the user to manually adjust the orientation and position of PDB models in the 3D workspace is important for the construction of multi-component simulations. Although forcefield simulation is generally required to produce realistic trajectories and docking interactions, a manual capability for inserting and arranging the relative locations of molecules is important for defining the initial conditions of a simulation and for many other kinds of structural modeling situations.

Until 3D input devices become ubiquitous, the mouse, in conjunction with keyboard-driven movement modes, can be used to effectively rotate and translate models in all 6 degrees of freedom. Movement of the mouse in two directions is combined with one of four mouse movement modes to produce a transformation. The four modes are rotate X-Y, rotate Z, translate X-Y, and translate Z, where X, Y, and Z refer to the view frame coordinate axes. For example, in the 'rotate X-Y' mode, moving the mouse up and down on the screen would result in a rotation about the X axis of the view frame. Moving the mouse left and right on the screen would result in a rotation about the Y axis of the view frame.

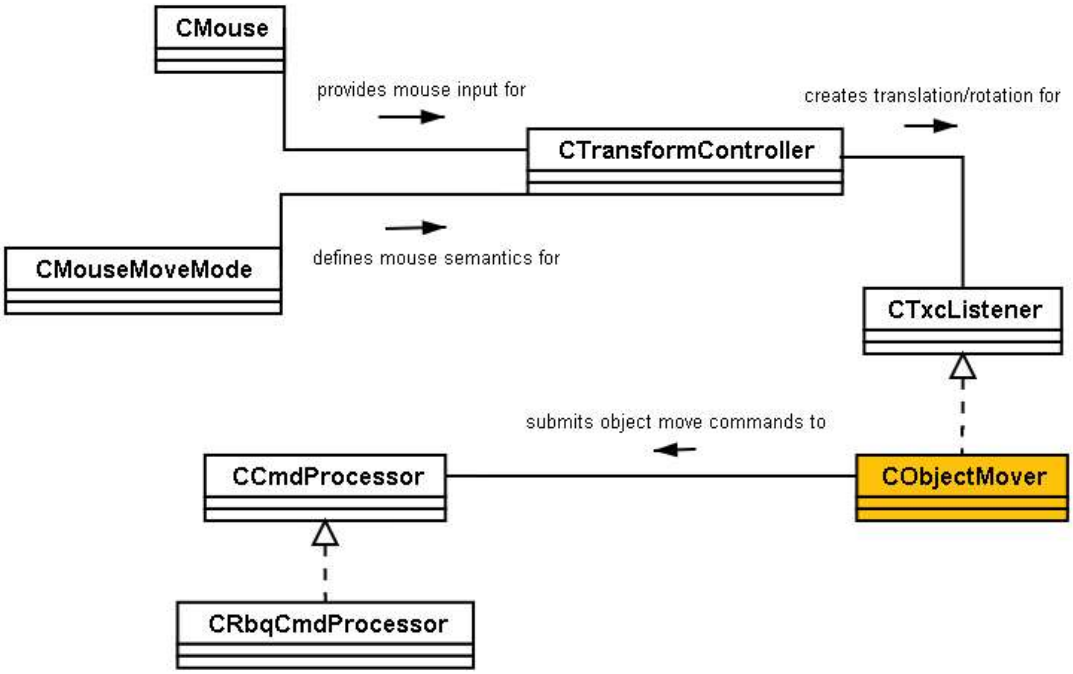

Fig. 23. UML diagram of CObjectMover and related classes.

The production of rotations and translations from mouse mode and movements is done by the CTransformController class, shown in Figure 23. The controller sends a rotation or

translation output to the CTxcListener interface. The CObjectMover class implements this interface and generates the necessary text command to produce the required movement, and submits the command to the command processor interface.

13.3.2. Changing the View

Transformation of the view is accomplished by the same kinds of mouse movements and modes that are used to transform objects. One difference is that rotations and translations are sent to a CViewMover object, which then applies them directly to a CView object instead of through a command string, as show in Figure 24.

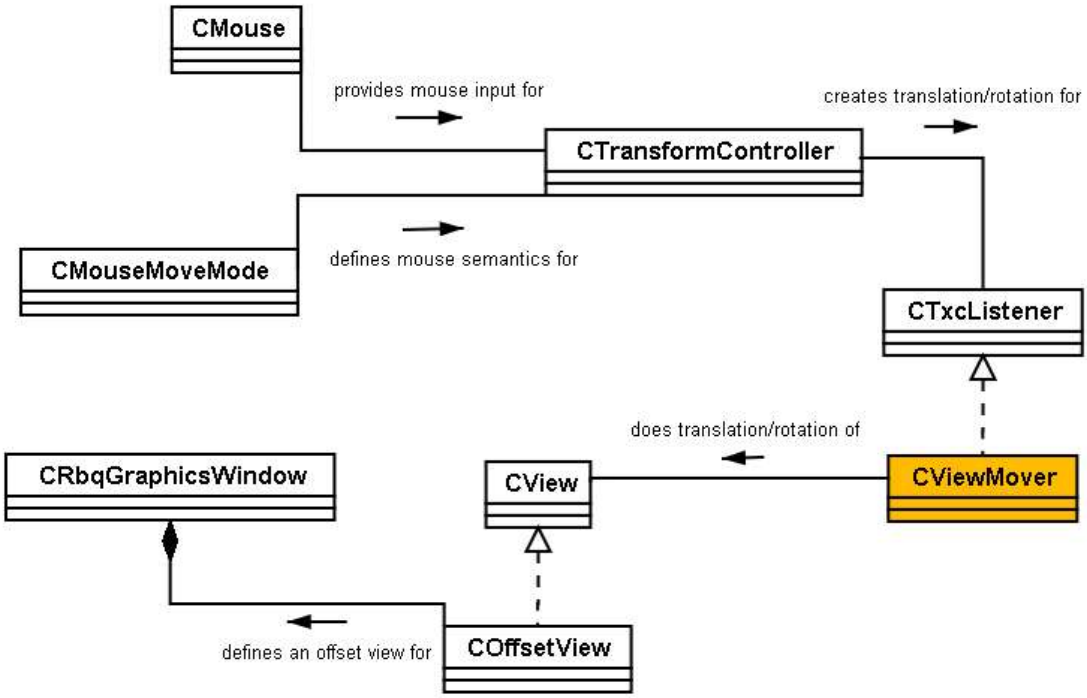

Fig. 24. UML diagram of CViewMover and related classes.

Although it would have been possible and more consistent to effect view changes through the command interface, at the time of implementation not all of the necessary commands

for view transformation had been implemented, and direct access of the view was considered a less serious violation of encapsulation rules in comparison to document objects.

There is one other important difference between object and view transformation. The view frame of reference has an inverse relationship with the world frame. During the development of viewing algorithms in the project, this inverse relationship was occasionally a source of confusion. The confusion was compounded by the requirement to have a view frame that could be placed at any position and orientation, with an offset rotation center that could be arbitrarily located. Many long hours of brutal combat with the OpenGL transformation matrices were needed before these requirements were satisfied in a clear and flexible way. For these reasons, it is worthwhile to discuss this issue in more detail.

By default, the camera viewpoint in OpenGL is centered at the origin of the world, looking in the -z direction. Consider a view transformation that would pull the viewpoint out along the +z axis, so that the origin is now visible. In the OpenGL scheme of things, there is no explicit 'view' object that can be directly transformed in this way. Instead, there is only a single 'ModelView' matrix to which all transformations are applied. The current state of this matrix determines the location of all objects that are subsequently drawn, until the matrix is further altered.

So, in order to cause the view to appear to be located at 10 units down the +z axis, it is actually necessary to translate the 'world', as represented by the Model View matrix, by 10 units in the opposite direction, causing the points of all objects drawn in the world to be shifted by that amount and direction. In addition to this simple inversion of translation,

there must almost be inversions of rotation. For example, as shown in Figure 25, if the view has been rotated by 90 degrees about the +y axis, so that it is now looking down the -x axis, and it is desired to tilt the view upward slightly, this would require a rotation about the +x axis in the view frame.

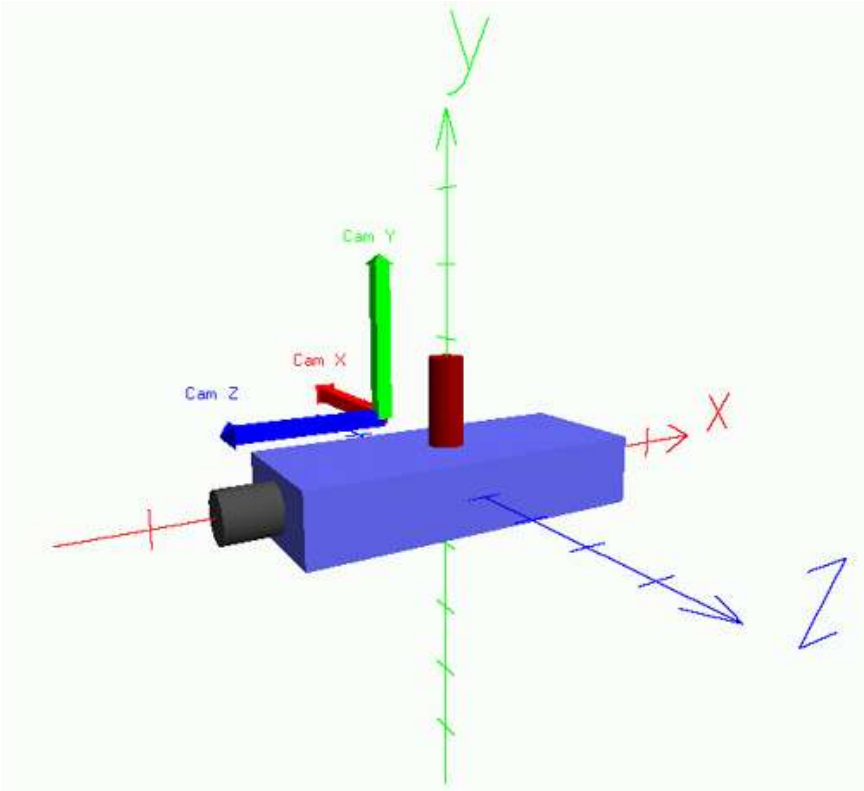

Fig. 25. Representation of a view coordinate frame that has been rotated by 90 ° with respect to the absolute coordinate frame.

However, this direction actually corresponds to the -z axis in the world frame, so there must be a means of converting a direction in the view frame to one in the world frame. This is done by calculating the inverse of the matrix that represents the current view. To calculate the inverse of an arbitrary matrix can be an expensive operation that is not always guaranteed to succeed. However, the rotation component of the 4x4 view homogeneous transform matrix is a 3x3 special orthogonal matrix, for which the inverse

is simply the transpose of the matrix (Gruber, 2000). Once the inverse of the view rotation matrix has been obtained, a direction in the view frame, represented by a vector, can be converted into a direction in the world frame by multiplying the view-frame vector by the inverse of the view rotation matrix.

Once we are able to convert vectors from view frame to world frame, there is a simple overall strategy for altering and applying the view. First, define a 4x4 homogeneous transform matrix to represent the view. Then, to translate or rotate the view by some vector in the view frame, first convert the vector to the world frame. Then translate or rotate the view homogeneous matrix by this world vector. Once all alterations of the view have been done, the application of the view is as follows. At the beginning of each drawing cycle, assign the view matrix to the OpenGL ModelView matrix. Then apply subsequent translations and rotations to the ModelView matrix, as needed, to locate objects in the scene, before drawing them. The CSimpleView class in the frameWk library is used to implement this strategy for defining the view.

13.3.3. Selecting Objects

There are a number of different ways to select an object. One very natural way is to use the mouse to click on the 2D image that represents the object. In the Ribosome Builder program, this capability is enabled by a powerful feature of the OpenGL graphics library. This feature, called 'picking', is a kind of reverse mapping that involves specifying a pixel from the final rendered 2D image, and then identifying the 3D geometrical object that produced that pixel.

Selection of objects uses a number of different modes, including what level should be selected in the object hierarchy (residue, chain, or entire PDB model), and whether the previous selection should be cleared or not. The CMouseSelectMode object supplies this information to a CSelectionController object, as shown in Figure 26. This information, in conjunction with a pixel coordinate from the CMouse object, is used to initiate a special 'selection mode' rendering of all objects drawn in the graphics window, which is done by the CRbqGraphicsWindow class. During this rendering, each drawn object pushes a numerical value onto a special OpenGL stack, where the numerical value corresponds to the index position of the object in the parent object's child list. The result is a sequence of numerical values that can be used to identify an object at any level in the PDB model object hierarchy.

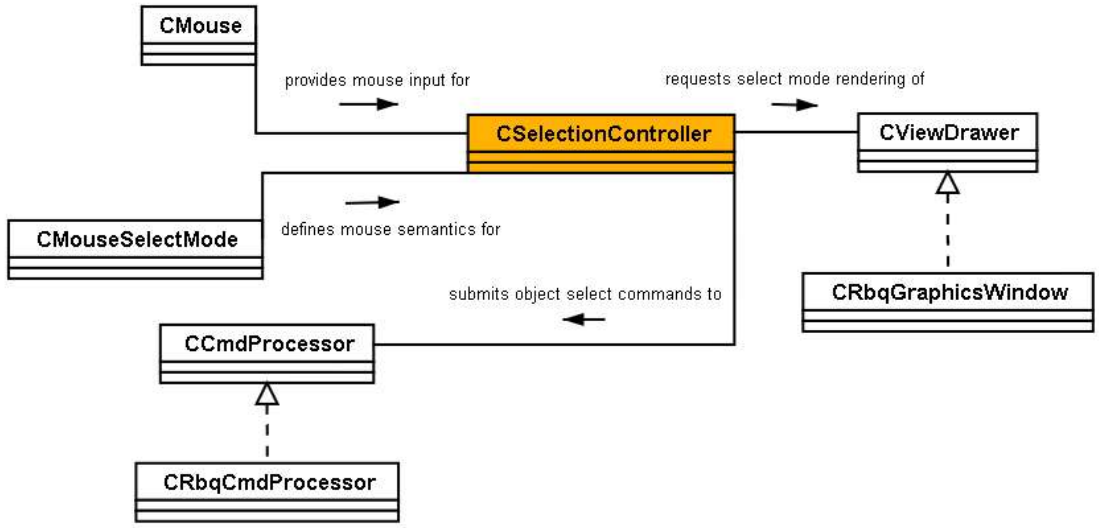

Fig. 26. UML diagram of CSelectionController and related classes.

Subsequently, if an OpenGL hit record is returned, it will contains the sequence of numerical values associated with an object that intersected the mouse pixel location. This

numerical sequence is then used to build a selection command string that is sent to the CCmdProcessor interface to select the object.

Chapter 14

Main Application Architecture

14.1. Introduction

The main application of the Ribosome Builder project is a program, 'rbuilder.exe', that serves as an integrated development environment for molecular modeling and molecular dynamics investigations of the ribosome and other macromolecular systems.

A screenshot of the program is shown in Figure 27. The main window shows four graphical user-interface components. The menus and toolbars are visible at the top of the main window, the 3D graphics window is located in the center left region, the PDB hierarchy window is at center right, and a status window is at the bottom.

The first version of the program implemented the graphical user interface with glut, an OpenGL toolkit (Robins, 2001). However, the functionality available from glut was limited, and so the second version of the program was implemented with a more powerful application framework, the Qt library (Trolltech, 2004). Although Qt is a cross-platform library that is well-designed and documented, it is not entirely free or Open Source. The

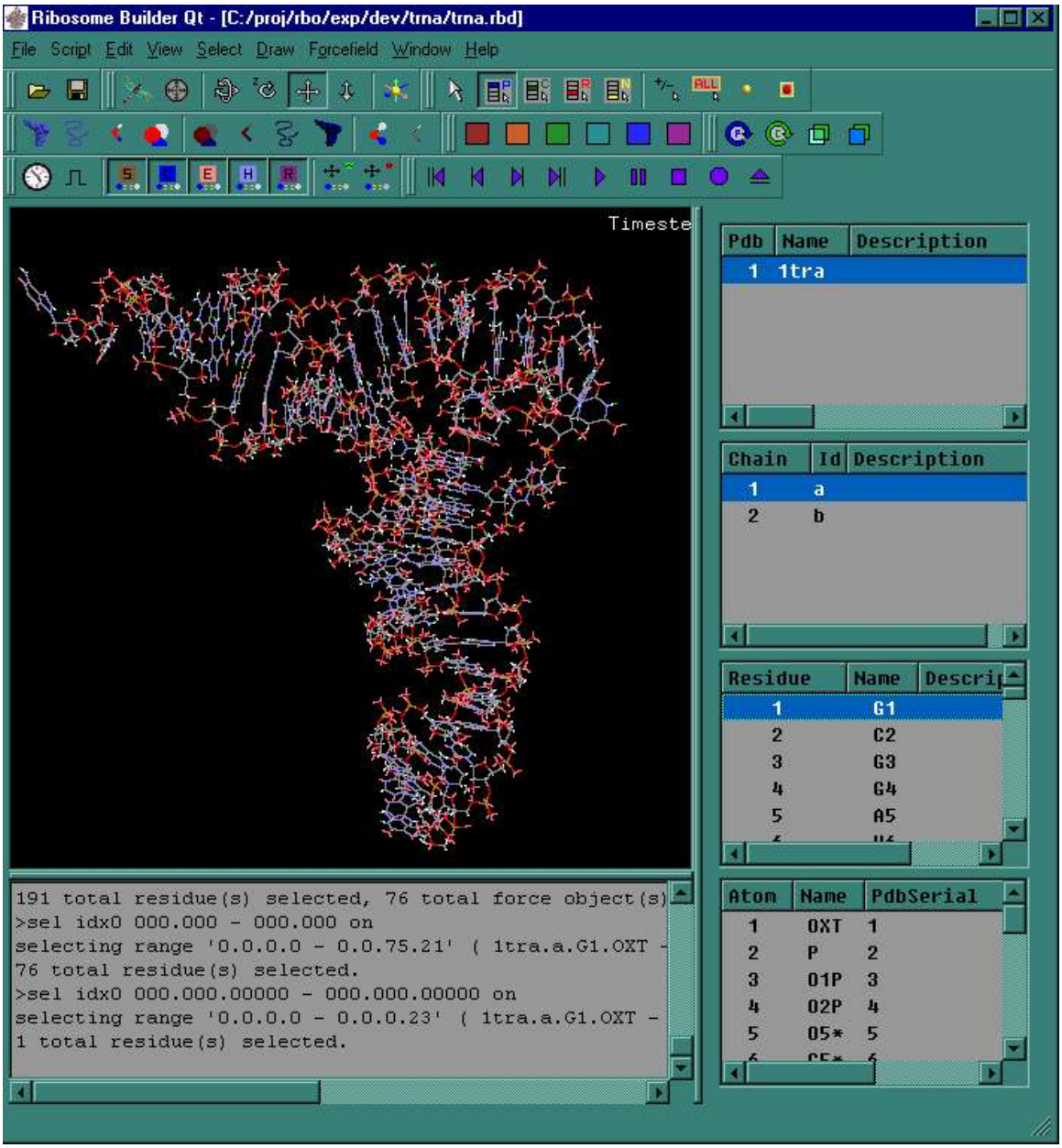

Fig. 27. Screenshot of application main window.

MS Windows version that was used for the Ribosome Builder program, Qt 2.2, is free for non-commercial use. However, it is an older version that is no longer supported, and is tied to the Microsoft Visual C++ 6.0 compiler. This poses a problem for collaborative development of the Ribosome Builder project, and so a subsequent version will be

created using a completely free and open Source toolkit such as wxWidgets (wxWidgets, 2004), or perhaps Qt 4.0 when it is released, as it has been promised to be open source. The graphical interface organizes a number of display surfaces in order to communicate relevant information to the user and to allow efficient control of the application. The display surfaces are referred to as 'widgets', a term used in the design of the XWindow system, the predominate windowing system on Unix operating systems. A widget is an object that provides a user-interface abstraction (XFree86.org widgets, 2004). The implementation of graphical interface widgets is done with components from the Qt framework, which defines a C++ class hierarchy of graphical objects. The base class for widgets is QWidget. A number of derived classes are explicitly defined during the creation of the main window of the program.

CRbqMainWindow is the top-level graphical widget and also functions as the top-level object for the entire application. It is the application-specific class that inherits from QMainWindow, a base class in the Qt library. The relationship between CRbqMainWindow and its principal component classes is shown in Figure 28. The discussion of the main application architecture will be organized into three general areas: graphical user interface and control, data representation, and the forcefield. In the figure, the graphical user interface and control classes are shown in green, the data classes are shown in blue, and the forcefield classes are shown in red.

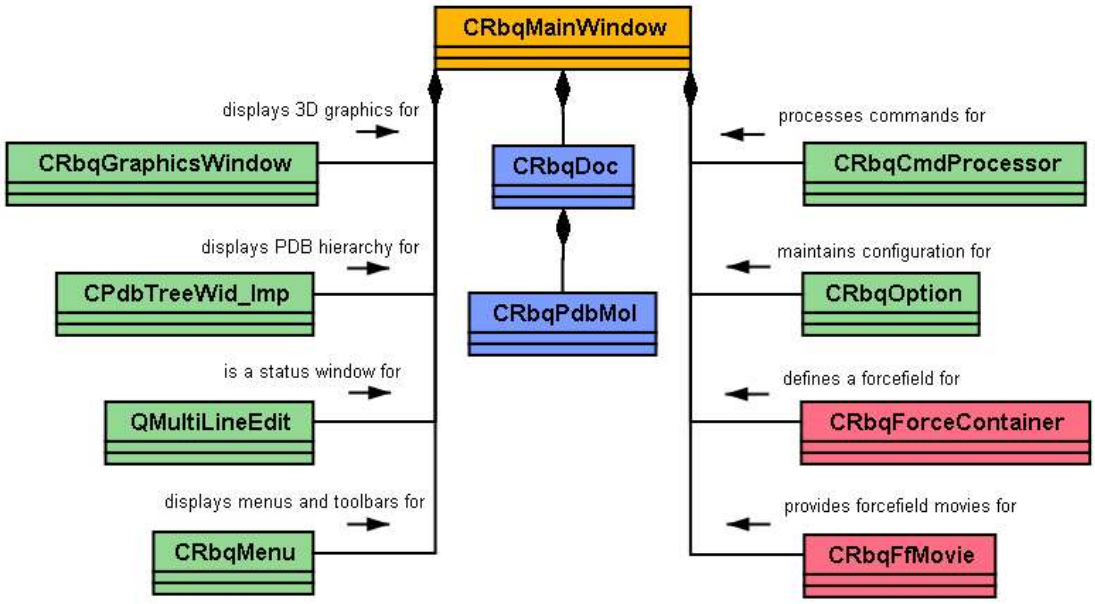

Fig. 28. Structural relationship between the top-level application object CRbqMainWindow and its principal component classes. GUI Control classes are shown in green, document and data classes are shown in blue, and forcefield classes are shown in red.

14.2. Graphical User-Interface and Control Subsystems

The term 'graphics' is intended here to refer to the graphical display of information in a software program. The term 'control' is intended to refer to the mechanisms that allow the user of a program to direct its operation. In software programs that employ a graphical user interface (GUI), these two distinct concepts are often closely intertwined. In describing the implementation of the Ribosome Builder main application, the software classes and objects that relate to these two concepts will be described under the common category of 'graphical user-interface and control subsystems'. The implementation is presented in two layers. The first layer consists of the core graphics system, which is dedicated to displaying the fundamental data of the application. The second layer consists

the user interface and control subsystems that surround and support the core graphics layer.

14.2.1. Core Graphics Subsystem

The purpose of the Core Graphics System is to display the PDB model data and supplemental annotations within a 3D space. This is done by the CRbqGraphicsWindow class, which defines an OpenGL rendering context. The rendering context defines the type of projection, lighting, materials and other configuration needed for display of a 3D scene by OpenGL. The content of the scene is input to the OpenGL display buffer from two main sources, as shown in Figure 29. The first source is the PDB model data, contained in CRbqPdbMol objects. The second source consists of optional annotations that can be displayed, which provide additional descriptions and information about the model data.

There are two types of annotations. The first type consist of 'built-in' graphic primitives and show things like the world and rotation axes, forcefield volume boundaries and selection boxes. A number of the primitives for drawing these annotations are defined in the rbqUtil library. The second type are produced by script objects, which are persistent entities of code that can be defined by the user. Within the draw() event of a script object, the user can call upon low-level drawing commands to produce whatever kinds of graphics are desired, as discussed in more detail in the section on script objects below.

The PDB model data and annotations are organized into a graphic object hierarchy, with a root graphic object at the top of the hierarchy. This facilitates uniform access and control for all displayed information. The graphic object hierarchy uses a base class,

CWkGraphicObject, to serve as a common interface for graphic operations that can be implemented by many different kinds of derived of classes.

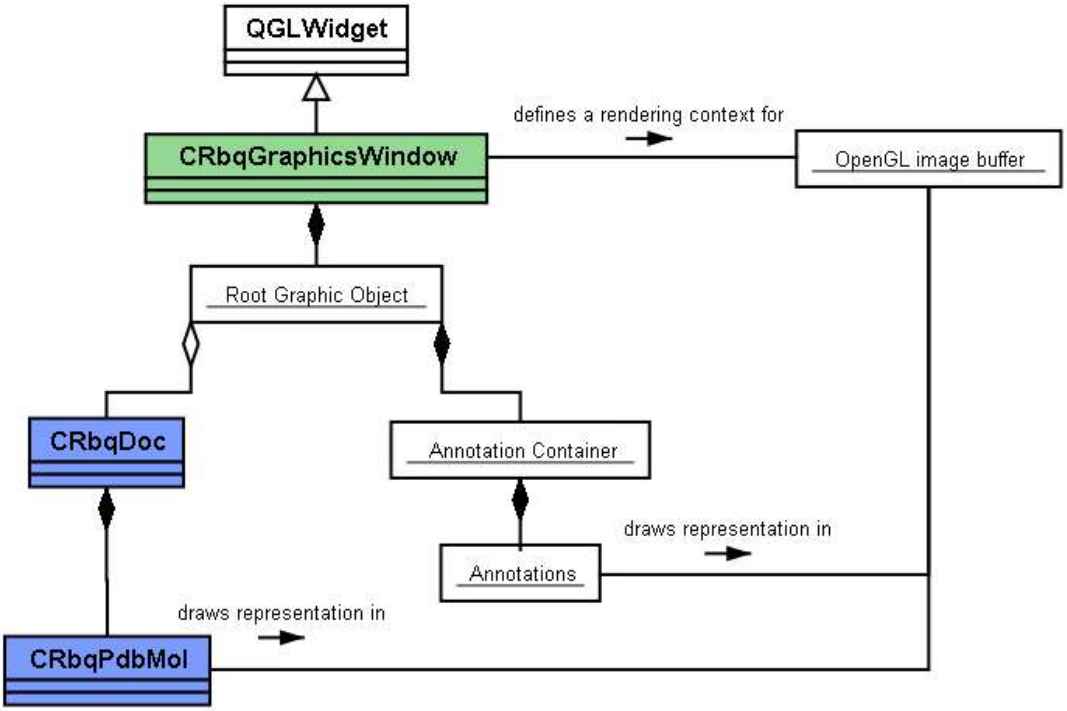

Fig. 29. Organization of the core graphics system, consisting of an OpenGL rendering context, PDB model data and annotations.

14.2.2. User-Interface and Control Subsystem

In a complex program with many different features and kinds of information, there is typically no single 'best' way to control the operation of the program. The most convenient method for input often depends on the particular task at hand and personal inclinations of the user. Consequently, it is important to give the user a number of alternatives for interacting with the program. The Ribosome Builder program provides menus, toolbars, command prompt, list-view controls, mouse movements, keyboard

accelerators and script files as input methods for the user. The classes in Figure 30 show the flow of control for some of these input methods. CPdbTreeWid_Imp, CRbqMenu and CInputCmdDlg_Imp are classes for generating text commands for interpretation by CRbqCmdProcessor, which then applies them to change the state of the Core Graphics System.

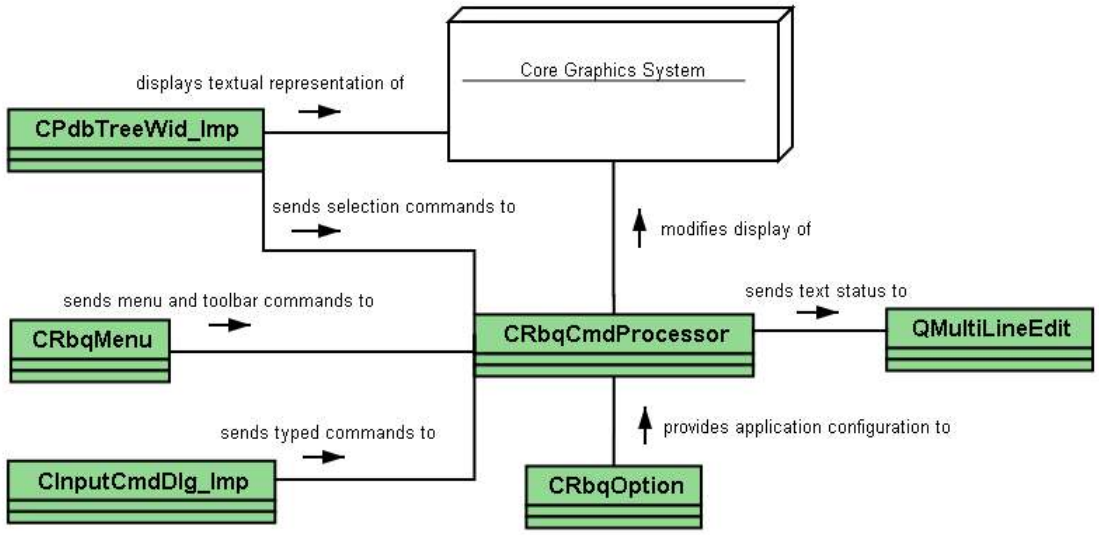

Fig. 30. Graphical user interface and control subsystem. It is composed of a core graphics layer surrounded by a second layer of graphical interface and control components.

14.2.2.1. CPdbTreeWid_Imp

CPdbTreeWid_Imp implements the PDB Hierarchy widget shown in Figure 27. This widget consists of a set of linked list-view controls that represent the structure of PDB models in the currently loaded document. Each list-view displays a sequence of items for a particular level of the hierarchy. If a single item is selected, then the components of that item will be displayed in the list-view for the level below. When multiple PDB models are loaded in a document, this control is a useful tool for quickly identifying and

displaying a particular subset of molecular objects from a global collection that may contain thousands of residues and atoms.

14.2.2.2. CRbqMenu

Menus are a common graphical user interface mechanism. They are valuable for organizing a large number of commands into a conceptual hierarchy that helps the user to locate a desired command. There are currently more than 150 menu items defined for the program, not including the customizable items in the Script menu. The Qt library has a flexible system for defining menu groups, menu items and graphical indicators of menu state. The CRbqMenu class makes use of this system to define and manage the hierarchy of menus for the Ribosome Builder program. Each menu item is associated with a text command string that is sent to the application command processor when the menu item is clicked.

The Script menu, one of the top-level menus, is a customizable hierarchy of submenus. It has menu items that load and display ribosome-specific models and features as well as multiple categories of other operations. The customization is useful because it enables convenient re-organization of operations into different groups as the program's capabilities and the user's needs change over time.

Although a hierarchy of menus is a useful interface mechanism, the traversal of the hierarchy can be tedious for deeply-nested and frequently used commands. Toolbars address this problem by grouping small buttons for related operations and continuously displaying them in the workspace, so that they are immediately accessible from a single mouse click. Each button is drawn with a graphical icon that ideally communicates the

purpose of the command in a succinct way to the user. In the program, there are currently 11 different toolbars which may be selectively hidden or shown, depending on the categories of operations needed by the user.

14.2.2.3. CInputCmdDlg_Imp

Menus and toolbars are especially well suited to the novice or casual user because the explicit display of commands gives a helpful starting point and context. However, the finite nature of screen real-estate often dictates that only a commonly-used fraction of the total command set will be represented in menus and toolbars. In the Ribosome Builder program, this fraction is currently around 60%. The command-prompt dialog, as implemented by the CInputCmdDlg_Imp class, enables access to the complete set of internally-defined commands, as well as the execution of externally-defined scripts.

14.2.2.4. CRbqCmdProcessor

The CRbqCmdProcessor is a central component in the application. Almost all user-directed actions to control the application will flow through the command processor. The processor works in conjunction with a set of command definitions, as previously described in the section on the wkParse library. When a text string specifying a command is submitted, it is validated and then executed. Status about the results of the command may be displayed in an associated status window, implemented by a QMultiLineEdit widget.

14.2.3. Library Connections to the Main Application

Figure 31 shows the connections from the low-level and application-specific libraries to the graphical interface and control components of the main application. Each of the

blocks in the figure has been discussed previously, and the interconnection between them gives a high-level overview for this portion of the system.

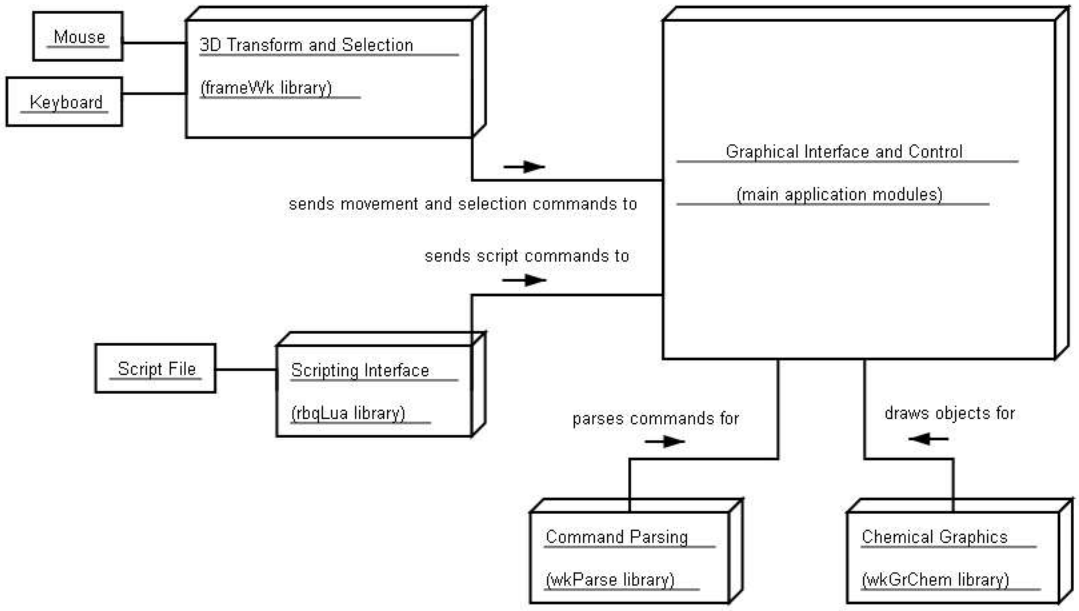

Fig. 31. Relationship between the project libraries and the main application, for graphical interface and control services.

The content of the Graphical Interface and Control box is shown in more detail in Figure 30. The content of the 3D Transform and Selection box is shown in more detail in Figure 23, Figure 24 and Figure 26. The content of the Command Parsing box is shown in more detail in Figure 22. The content of the Chemical Graphics box is shown in more detail in Figure 16 and Figure 20.

14.3. HTML Browser Interface

14.3.1. Motivation

The primary user interface in the main program is supplemented by an additional interface using an HTML browser such as Mozilla. As a standalone application, the Ribosome Builder program can support 3D graphical operations more easily and flexibly than a third-party HTML browser. However, the browser can provide powerful features that complement the 3D graphical environment.

Modern browsers are highly optimized to display multiple tabbed 2D windows of hyperlinked text, tabular data and graphics in a universal open standard. Furthermore, the basics of HTML are simple enough so that most users can create small, organizational HTML documents. These documents, in conjunction with a suitable interface to the 3D graphical application, can be a powerful way of navigating and working with molecular modeling data that is spread across many different directories and file types.

When using a browser with the Ribosome Builder program, the windows of these two applications are typically positioned next to one another without overlapping and are sized to take advantage of the full screen space. Future deployment of multiple monitor configurations on desktop systems will make such linked application use even more convenient.

The browser is typically loaded with an organizational HTML document that contains a set of hyperlinks. When a hyperlink is clicked, it causes a command to be sent from the browser to the Ribosome Builder to execute a particular script. In this way, a set of defined hyperlinks and associated scripts can allow the user to load models, adjust the graphical representation and initiate a suite of molecular dynamics setup operations with just a few clicks on a particular HTML page. The user can then navigate to another

HTML page in a different directory, which may define an entirely different modeling project, and initiate another set of operations in the same way.

This control strategy combines the benefits of the familiar browser interface, as exemplified in Protein Explorer and Chime (Martz, 2002), with the power and flexibility of a scripting language embedded in a standalone open source application such as PyMOL (DeLano, 2002).

14.3.2. Implementation

Because the main program is currently constructed as a single-threaded application, and listening for html browser requests on a network socket would suspend thread execution until a request comes in, a separate application was created to run as an independent process to handle the socket communication, as shown in Figure 32.

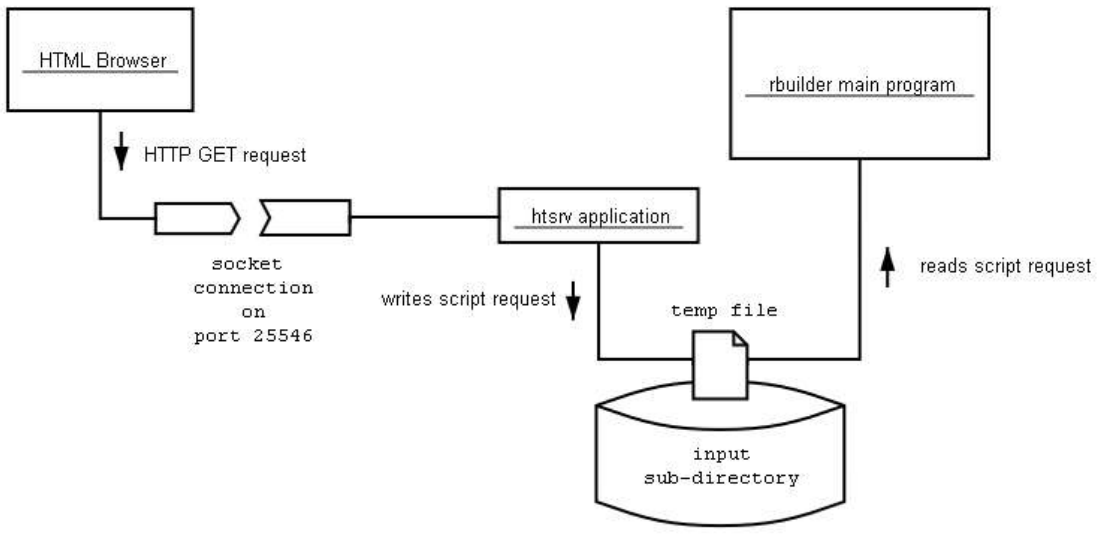

Fig. 32. Interface between an external HTML browser and the Ribosome Builder program.

The HTML request is encoded using the CGI parameter specification (Guelich et al., 2000). In order to demonstrate the format used, the HTML page from the browser communication test application will be given as an example. This sample page, located in the 'exp/dev/browserTest' subdirectory, is included in the program distribution for the purposes of testing the web browser interface.

The directory contains three HTML files that define a simple web page with two frames, consisting of a main window and a small status window. A sample image of a web browser displaying a similar page is shown in Figure 44. The purpose of the status window is to inform the user of the status of the GET request without affecting the main browser frame. The browser test web page contains three hyperlinks, with HTML code as shown below:

```
<ul>
<li><a target="status"
href="http://localhost:25546/?scriptPath=exp/dev/browserTest">Set
script path</a>
<li><a target="status"
href="http://localhost:25546/?scriptFile=loadTrna.lua">Load tRNA
model</a>
<li><a target="status"
href="http://localhost:25546/?scriptFile=rotView.lua&90%200%201%
200">Rotate view</a>
</ul>
```

The first link defines a request to set a default path for script files. This allows the subsequent links in the page to specify only the file name of a script instead of the full path. This is helpful if the location of the page is subsequently moved, so that the path only needs to be updated in a single location rather than for every hyperlink.

The next two links each request the execution of a script file. The first link executes the script without any additional arguments. The second script is executed and receives a

single argument, which is the string '90 0 1 0', where the spaces in the string are encoded with the token string '%20'.

The low-level socket communication of data is handled by classes in the FPSocket library, discussed previously.

When the htsrv application receives the GET request from a browser, it performs CGI decoding on the string and then places it in a temporary file in an input directory that is periodically scanned by the main application. The main application will open the temporary file, read the request and attempt to execute it. It will then delete the temporary file from the input directory. This rather cumbersome scheme has been required for the current single-threaded implementation, but it has worked well in practice.

14.4. Documents

In the Ribosome Builder program, a document is the container that holds the data used for modeling and simulation work. This data can include static structural models loaded from PDB files, geometrical and graphical attributes, dynamic segment definitions, and application state information.

The application is designed to work with only a single document at any one time. A new document can be created from within the application, or it can be created from a pre-existing file. There are two types of files that are used to create documents. The first is a Protein Data Bank (PDB) file, which is a common format for atomic structural data. The second type is a Ribosome Builder Document (RBD) file, which is created by the program.

When a PDB file is opened, its first model becomes the current document in the Ribosome Builder program, replacing any previous model. The model is shown in a default position and graphical state. The user can work with this single model, then close the document and load some other PDB file.

However, the user is not limited to working with a single PDB model. It is also possible to insert additional PDB models from other files into the current document. This allows the user to create a multi-model configuration in which the spatial positions and orientations of the models can be adjusted. It is also common to change the graphical appearance of the models in a way that relates to the purpose of the configuration. These changes can then be saved as an RBD file, allowing the configuration to be restored at some subsequent time.

If any dynamic segments have been created, the information needed to re-create them will also be saved to the RBD file. Dynamic segments are objects that represent a subset of the static PDB models of the document, and are used in molecular dynamic simulations. The RBD document also contains some application state information that is normally associated with a document, such as the current viewpoint and window background color.

14.4.1. RBD File Format

Because of the hierarchical nature of the model data, the graphical and dynamic information is saved in an RBD file using a simple XML-like format. The format is simple because it lacks a formal Document Type Definition (DTD), which is a set of declared rules for how the document tags are parsed in XML (Ray, 2001). The data is

simply embedded between start and end tags with various names that correspond to object classes and their components.

One feature of the file format that is notable is a 'client data' section that enables future version modifications and extensions to saved data without breaking backwards-compatibility with previous versions of the file format. The client data section is saved in a separate section of the file, prior to the document model data. The 'document save' code can store arbitrary strings in the client data section, with the content bracketed by a custom tag name. When the file is subsequently read, the client data is read in a uniform way and stored in a list. The 'document read' code can optionally retrieve a chunk of client data during file processing. This scheme allows an older version of the program to read a newer file format, because the extension data is in the client data section and will not be seen by the document class parsing code. Likewise, a newer version of the program can read an older file format version, because the model data is still in the same format, and the missing extension data can be ignored if desired.

Another valuable feature of the file format is a form of 'run-length-encoding' that is used to efficiently record the graphical state of model objects. For large models with thousands of residues, often only a small fraction of the residues have their graphical state adjusted in a unique way. If the graphical state of a continuous sequence of residues is the same, a single chunk of data is saved for that sequence. A new chunk is then created for the next residue to be saved, and will also represent subsequent residues sharing that state, if any. This commonly avoids saving thousands chunks of data, which would be required if the graphical state of each residue was saved individually.

14.4.2. CRbqDoc

CRbqDoc is the class that defines the document object. The structural relationship between the document classes and the rest of the application is shown in Figure 33. The principal component of CRbqDoc is the CRbqPdbMol class, which defines objects to hold the structural, dynamic and graphical data for PDB models. A separate CRbqPdbMol object is created for each PDB model loaded into the document.

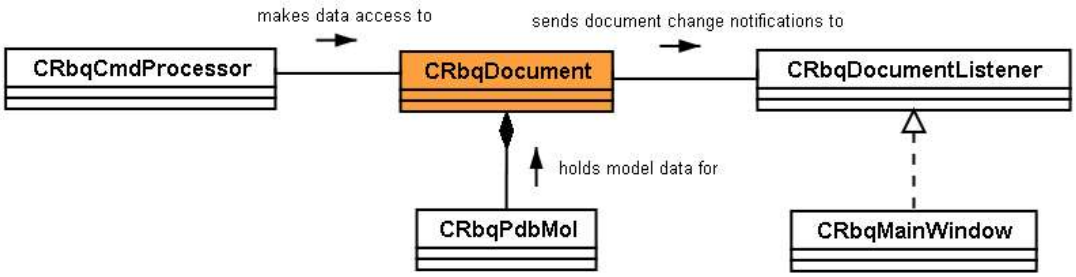

Fig. 33. UML diagram of CRbqDocument and related classes.

Actions upon the document are produced by the CRbqCmdProcessor, and the resulting changes in document state are communicated to the application through the CRbqDocumentListener interface. One example would be a notification that is sent when document contents have changed, indicating a need to save. Another example is a message sent when the document selection state changes, requiring updates to other parts of the application that are associated with document selection, such as the Pdb hierarchy window. The document listener interface is an important mechanism for isolating the document from other parts of the application. This isolation is beneficial for producing a clean implementation of the document class, because it allows the implementation to be focused on the single task of representing and providing access to document data.

14.4.3. CRbqPdbMol

CRbqPdbMol is the class that maintains the fundamental model data for the application. The data is divided into two distinct types: the static model created from a PDB file, and optional dynamic models that can be created from all or part of the static model.

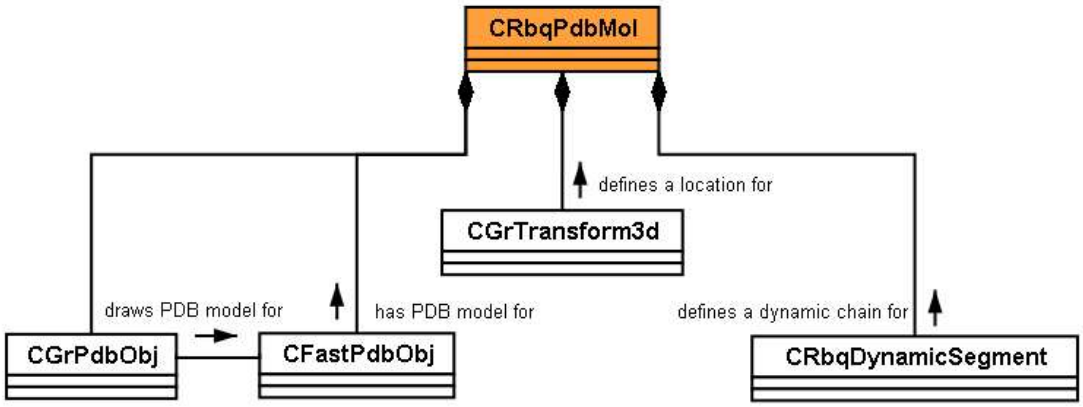

Fig. 34. UML diagram of CRbqPdbMol and related classes.

As show in Figure 34, the static model is represented by a CFastPdbObj object, discussed previously in the section on the wkFastPdb library. The graphical representation is produced by an associated CGrPdbObj object. The CGrTransform3d object is a geometrical transform that holds changes in position and orientation that have been made by the user. These changes are applied to the static component, resulting in the collective rotation and translation of all atoms in the PDB model.

When a dynamic segment is created, it 'inherits' the current transformation of the static model, but any subsequent transformations will occur independently. A dynamic segment is created from a selected sequence of residues in the static model. Each dynamic segment is represented by a CRbqDynamicSegment object. The internal structure of a

dynamic segment is more complex than the static component, and is discussed in more detail in the forcefield section below.

14.4.4. Linkage of Static and Dynamic Models

Because dynamic segments are components within a CRbqPdbMol object, they live and die with that object. An alternative would be to maintain dynamic models in a separate collection from static models. A number of reasons motivated the current design. These include the close conceptual linkage of dynamic models to their static source, reliance on static models for some of the structural data, and homogeneity of data at the document level. This may not have been the best choice however, and a future reorganization of dynamic data may be result in a cleaner implementation.

14.4.5. Specification of Model Data for External Access

As mentioned previously in the section on wkChem library development, there is a need for well-designed mechanisms to access data embedded within an object hierarchy. Both static and dynamic models exist as a hierarchy or 'tree' of objects, where a model consists of multiple chains which in turn consist of multiple residues which in turn are composed of multiple atoms. In many places in the main application, this data is accessed using a scheme that avoids the need for direct memory pointers to model objects.

The scheme works by using a sequence of numbers that correspond to the positioning of an object within its parent containers. For example, the first atom of the tenth residue of the third chain of the fifth pdb model would be specified by the numerical sequence { 4, 2, 9, 0 }. Note the use of a 0-based index, where the first position in a container is

denoted by the value '0'. The term 'index position' will be used to refer this composite set of numbers (numerical sequence).

This scheme has a number of advantages. It is natural for iterating the full tree or subtrees of objects, using nested 'for' loops. It can be used to specify a range of objects. This is done with a pair of index positions that mark the endpoints of the range. The validation of an index position or range is straightforward. Index positions can be passed as function parameters in a variety of formats. They can be passed as individual numbers, as an array of numbers in a single integer list parameter, or converted to a string representation where they are separated by a '.' character:

```
GetObjectMass( 4, 2, 9, 0 );
GetObjectMass( IntegerList( 4, 2, 9, 0 ) );
GetObjectMass( "4.2.9.0" );
GetObjectMass( "1ffk.A.GLY10.CA" );
```

In the final item in the list above, the numerical values have been converted into the corresponding names of the objects. This final form is also suitable for interactive command-line queries submitted by a user. A number of routines have been created for interconversion between this 'dotted-name' specification and the index position form.

14.4.6. Selection

When the user asks the program to perform some operation on a set of objects, it is necessary to somehow identify which objects are the intended targets. One possibility is to require the user to explicitly specify the target objects for each and every operation. However, in many circumstances, it is more convenient to use the concept of 'selection'. The act of selection is defined here as the addition of objects to a special group, called the

'current selection'. A number of operations in the program are defined to operate only upon objects in the current selection.

In the previous discussion of the graphical interface and control systems, a number of input mechanisms for object selection were mentioned. Ultimately, all of these inputs are translated into a selection command that is received and interpreted by the command processor, which then acts upon the document to adjust the current selection. The current selection is actually implemented by defining boolean variables and associating them with model objects. The current selection then consists of the set of all objects whose selected state is true. Currently, low-level objects, such as residues in a CFastPdbObj object, do not define a separate variable for selection state. Instead, the boolean variables are defined in the associated classes that are responsible for the graphical representations of these low-level objects. This has resulted in a number of constraints which may be eliminated in future implementations by the explicit allocation of a 'selected state' variable to the low-level object data.

There are a number of ways for communicating the selected state of objects to the user. The most common is to use a 'selected' color when drawing objects. Another graphical indicator is an axially-aligned wireframe bounding box that can be drawn around the current selection. The program can also output a textual report that lists the totals and names of selected objects.

14.5. Dynamic Models and the Forcefield

In this section, the creation, structure and use of dynamic models will be discussed. The primary use for dynamic models is the simulation of their interactions and movements

within a molecular dynamics forcefield. The overall architecture for the forcefield will be presented, followed by a detailed examination of the forces and torques that define it.

14.5.1. Dynamic Models

A dynamic model is distinguished from a static model by the ability to change the conformation of the dynamic molecule. This conformational change occurs through the rotation of atoms about their covalent bonds. In contrast, all the atoms of a static model maintain their positions relative to each other.

The representation of static models is designed for efficient use of space and processing time. This efficiency is important for working with ribosome models. In the program, multiple models of both the 30S and 50S subunits can be loaded simultaneously for comparative purposes, which entails the representation of hundreds of thousands of atoms. A consequence of this requirement for efficiency is that dynamic models are not created automatically when static models are loaded. Dynamic models only exist after being explicitly created by the user, and they use a different data representation from static models.

Dynamic models are created from sequences of static residues. The result is a subset of a linear chain of residues that is referred to as a 'dynamic segment'. For example, the entire length of a static chain can be converted into a single dynamic segment, or individual fragments of the chain can be selected and then converted to produce multiple dynamic segments. This capability for partial conversion is important for a number of reasons. First, in a model that has a very large chain, conversion of the entire chain to a dynamic model may impose a large cost on the processing resources of the system. A second

reason is that many static models have gaps in a chain of residues that result from unresolved residues in the structure determination. Finally, only a small fraction of PDB residue types are currently supported for conversion to a dynamic representation. These consist of the standard nucleic acid and amino acid types, along with a small number of heterogeneous types that are encountered in ribosome modeling. The total number of heterogeneous PDB residue types is greater than 4,600 (Bsm, 2004).

14.5.2. Generalized Forcefield Container

Before discussion of the detailed structure of the classes that implement dynamic models, it is worthwhile to consider them in the context of the operating forcefield. Molecular models can be simulated by forcefields in many different ways. In some docking simulations, a large component may be held rigid, and only the small ligand is modeled with flexibility (Halperin et al., 2002). In other simulations, all atoms are allowed to move. The range of possible representations and implementations motivated the development of a generalized container architecture for the Ribosome Builder forcefield. This generalization is demonstrated in Figure 35, where the CRbqForceContainer class acts as a container for simulating interactions between a collection of force objects. The CRbqForceObject interface makes it possible for different types of objects to be represented within the forcefield. The force container is responsible for directing the pairwise interactions between objects, but the actual calculation of forces is determined by the specific objects themselves. Currently, there is only a single object type that is used. This type is implemented by the CRbqNodeBsjForceObject class, which represents

an individual residue of a nucleic acid or protein chain. The structure and function of this class is discussed in more detail below.

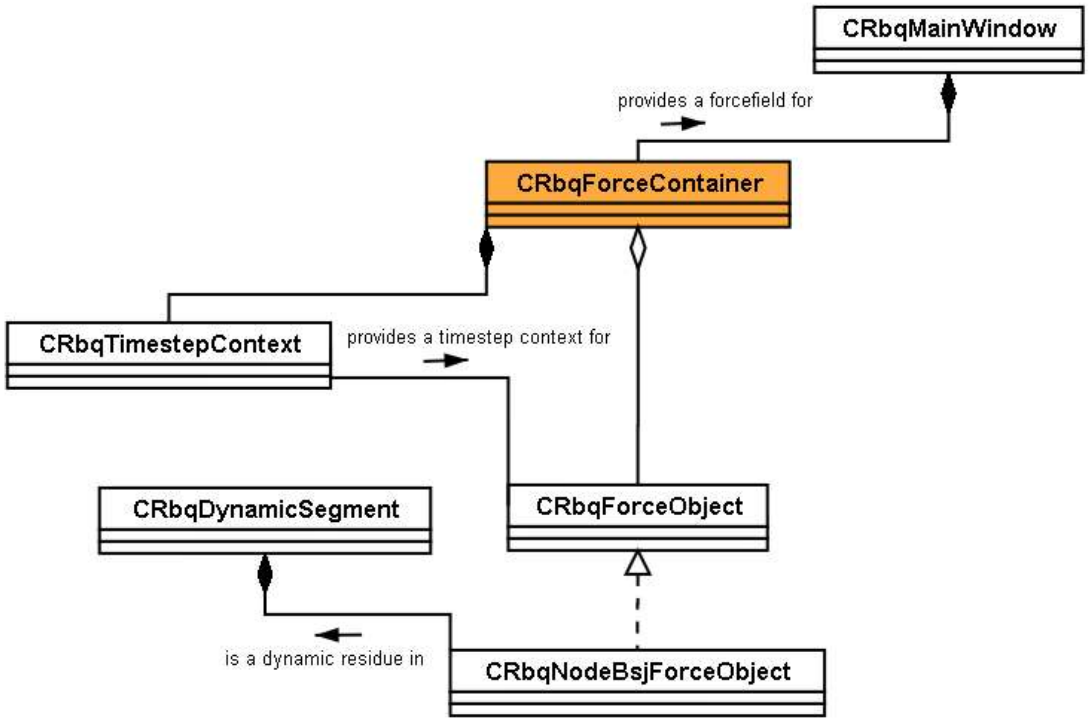

Fig. 35. UML diagram of CRbqForceContainer and related classes.

To simulate the activity of objects over time, the forcefield calculations are done in a sequence of discrete 'timesteps', where each timestep represents a small interval of real time. The timestep processing is divided into three distinct phases. The first phase is an initialization process, in which accumulated forces and torques from the previous timestep are cleared, along with any associated statistics and energy values. The second phase is where the actual calculation of forces and torques is done, as a result of interactions within and between the objects in the forcefield. The third phase is the

movement phase, in which objects are moved in space as a result of their mass and the total forces and torques acting upon them.

During the timestep, a number of external parameters can influence the calculated forces and movement. These parameters include the length of the timestep, the dimensions of the forcefield volume, the application of frictional forces, and many other parameters that are specific to a particular type of interaction, such as steric contact or hydrogen bonding. Many of these parameters may be dynamically adjusted by the user. The presentation of such parameters during the timestep is done with a CRbqTimestepContext object, which is passed as a parameter in the timestep functions.

14.5.3. Dynamic Model Structure

14.5.3.1. CRbqDynamicSegment

The internal structure of the CRbqDynamicSegment class is shown in Figure 36. This class acts as a container for multiple residues. Each of the residues within a dynamic segment object is defined using the four component objects shown in the figure. The core model data of the residue consists of the atoms and bonds contained within a CTorGraph object. This object provides a set of atoms and associated operations that allow the positions of the atoms to be changed as a result of rotation about their covalent bonds. The core geometrical data in a CTorGraph object is used for both graphical and forcefield purposes. As discussed previously, it is important to maintain a separation between these different purposes. The graphical representation of a dynamic residue is performed by the CGrTorGraph class, acting through the parent hierarchy of CRbqTorsionGraph and

CRbqDynamicSegment. The dynamic functions are implemented by the CRbqNodeBsjForceObject class.

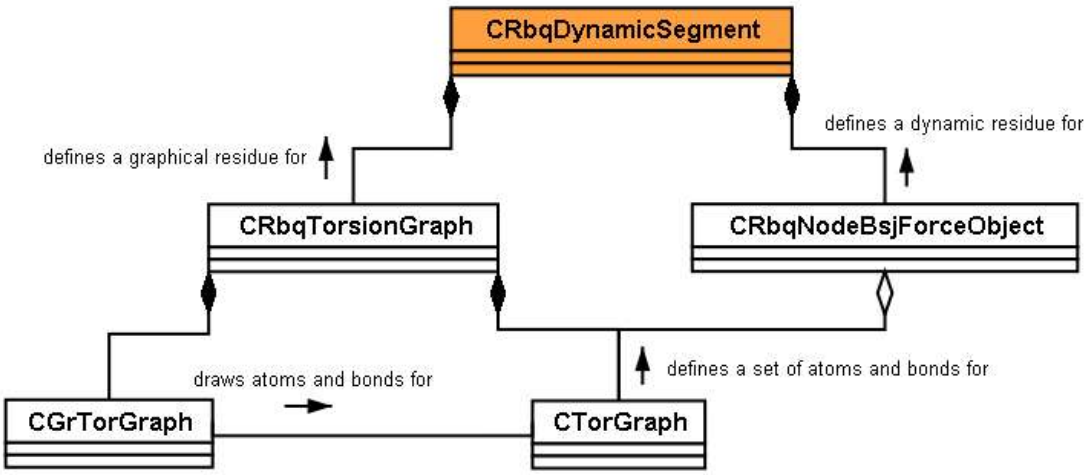

Fig. 36. UML diagram of CRbqDynamicSegment and related classes.

14.5.3.2. CRbqNodeBsjForceObject

CRbqNodeBsjForceObject represents a covalently-bonded set of atoms in the forcefield. The set of atoms are those from a single residue, which may be isolated or part of a linear chain. If the residue is a part of a chain, it may also be bonded to a previous and a next residue. The name of the class derives from two aspects. The 'Node' part refers to the fundamental elements within the object that receive forces, torques and undergo movement. Currently, each node corresponds to a single atom, but in previous versions multiple atoms could be consolidated into a single node. The 'Bsj' stands for 'Ball and Socket Joint', which was an early name for the method used to calculate the covalent bond forces. This method is discussed in further detail below, in the section on the 'Shadow Point' algorithm for covalent bonds.

The CRbqNodeBsjForceObject class is one of the more complex components in the project, as shown by its internal structure in Figure 37. The relationship with the CTorGraph and CGrTorGraph components was discussed previously, in the context of the parent CRbqDynamicSegment structure. The only purpose of the CGrTorGraph connection to the force object is to hold the selected state. The fundamental force data component is the NFData structure, which holds force and torque data for the nodes of the force object. In addition to vectors for accumulated force and torque, each NFData structure holds a CGrTransform3f object, which defines a position and orientation for the node in absolute space.

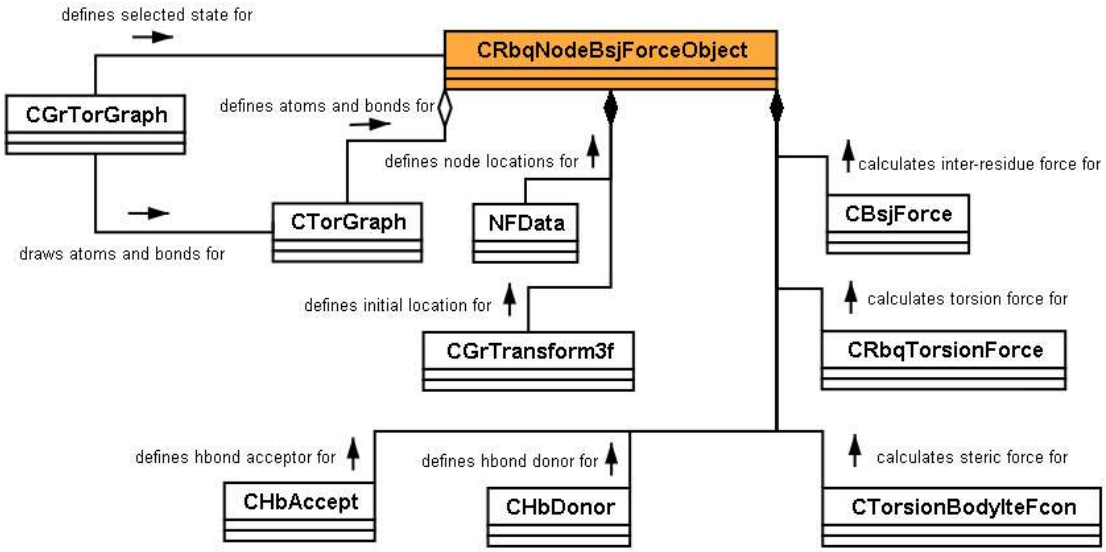

Fig. 37. UML diagram of CRbqNodeBsjForceObject and related classes.

There is a single CGrTransform3f object which defines the initial location for the entire force object. This transform may hold a modified location that was inherited from the originating static PDB model. After the atoms in the force object have undergone movement, the initial transform enables them to be restored to their original starting

positions. This allows the user to perform multiple simulations on the same set of force objects by periodically resetting them to their initial state.

The remaining five components in the figure are classes that are involved in calculating the different kinds of forces. CTorsionBodyIteFcon, a class from the wkEncon library, calculates steric interactions between intra-residue atoms. External, inter-residue steric forces are calculated by the CTorsionBodyTgaxFcon class, which is not shown because it is created transiently on the stack. CRbqTorsionForce is used to calculate both intra and inter-residue torsion interactions. CBsjForce is used to calculate inter-residue covalent bond interactions. The corresponding intra-residue bond calculations are done by member functions of CRbqNodeBsjForce object instead of by a dedicated component class. The detailed definitions of the force functions are described in the subsequent section.

14.5.4. Definition of Forcefield Interactions

The forcefield functions and parameters in the Ribosome Builder were developed from scratch, without reference to existing forcefields such as AMBER or CHARMM. This was an intentional design decision, in accord with the goal of modeling extremely large structures such as the ribosome. The initial emphasis has been on computational efficiency, as opposed to more accurate, but time-consuming, simulations with established forcefields.

The molecular dynamics forcefield calculates the interactions between atoms and updates their positions at each timestep. The forces and movements are not calibrated to a physical timescale, but they do maintain bond lengths, bond angles and steric exclusions.

Additional force functions are applied to reproduce electrostatic, hydrogen bond and base-stacking interactions.

The movement of each atom in the forcefield is calculated from explicit accumulations of forces and torques on the atom. This is in contrast to conventional forcefields, where the forces are calculated from a potential energy representation. Because an atom has an explicit orientation and valence structure, the interacting forces are used to compute torques that rotate the orientation, in addition to translating the atom center-of-mass.

14.5.4.1. Composite Force and Torque Functions

The total force and torque on an atom at each timestep are the sum of forces and torques from all atom pair interactions involving that atom, as given in Equation 9 and Equation 10. The force equation contains terms for steric (s), covalent (c), electrostatic (e), hydrogen-bond (h) and aromatic base-stacking forces (r). The torque equation contains terms for covalent (c), torsion-bond (t) and hydrogen-bond (h) torques.

$$F = F_s + F_c + F_e + F_h + F_r \tag{9}$$

$$T = T_c + T_t + T_h \tag{10}$$

The atom pair interaction for each term depends upon the type of atoms and threshold distances. For example, the steric interaction term is skipped for 1-2, 1-3 and 1-4 bonded atoms, and all steric interactions are skipped between atom pairs more distant than 25 A.

14.5.4.2. Steric Interaction

The steric interaction term is a Lennard-Jones function, as shown in Equation 11. k is a scaling factor, calculated from the sum of the van der Waals radii of the two atoms, which will produce a force of zero at the equilibrium distance. vij is the unit vector between atoms i and j.

$$F_s = \left(\frac{1}{(k \cdot d)^n} - \frac{1}{(k \cdot d)^m} \right) \cdot \hat{v}_{ij} \tag{11}$$

14.5.4.3. Covalent Bond Interaction

The covalent interaction is calculated from the shadow-point method, described in more detail below. Briefly, the method relies upon an set of covalently-bonded atoms with an initial geometry. As the atoms change position over time, a shadow point is the original location of one atom relative to the transformed frame of a second atom. The force on an atom i is the sum of its shadow-point force and the reaction of the shadow-point force of atom j, as shown in Equation 12. The corresponding torque is the sum of the atom i torque and the atom j reaction torque, shown in Equation 13.

$$F_c = F_{is} - F_{js} \tag{12}$$

$$T_c = T_{is} - T_{js} \tag{13}$$

The force between an atom and its shadow point is given by the spring force function in Equation 14. The torque produced is the cross-product of the moment arm and the applied force. For the direct torque, T_{is}, the moment arm is zero, because the point of application

of the force and the center-of-mass of the atom coincide. However, the reaction torque T_{js} will be non-zero for reaction forces applied at the shadow-point of atom j, as shown in Equation 15. The points Pi and Pj refer to the current locations of atom i and atom j. The points Qi and Qj are the shadow points for atoms i and j, and are calculated by multiplying the original location vector of the atom by the transform matrix of the other atom.

$$F_{is} = -k \cdot d_{is} \cdot \hat{v}_{P_i Q_i} \tag{14}$$

$$T_{js} = (Q_j - P_i) \otimes F_{js} \tag{15}$$

14.5.4.4. Torsion Bond Interaction

The torsion torque, Tt, is the torque produced on an atom involved in a torsion bond, consisting of four atoms A, B, C and D. The torque is derived from the steric force between atoms A and D. Figure 38 shows this steric force as F_S, along with its axial and tangential components F_r and F_t.

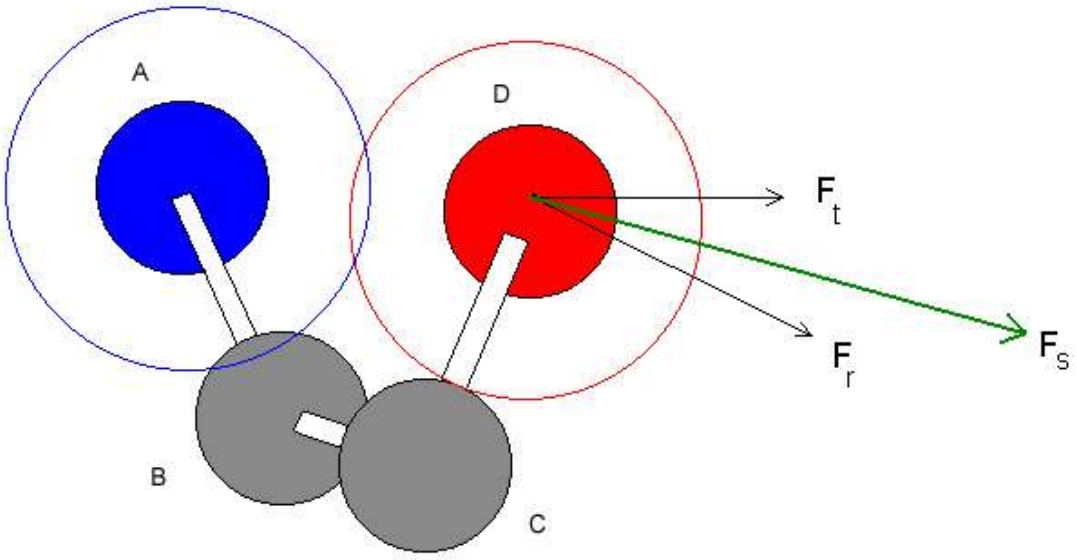

Fig. 38. Forces used to calculate the torque in a torsion bond. Atoms A, B, C and D compose the torsion bond. F_S is the steric force exerted on atom D by atom A and F_r and F_t are its axial and tangential components.

The magnitude of the steric force exerted on atom D by atom A is given by Equation 16. F_r is determined by projecting the steric force along the bond vector, as given by Equation 17. F_t is calculated as the difference of these two, as given by Equation 18. F_t is then used to produce the torsion torques on atoms B and C, as shown in Equation 19 and Equation 20.

$$F_{AD} = \frac{1}{(k \cdot d_{AD})^n} - \frac{1}{(k \cdot d_{AD})^m} \tag{16}$$

$$F_r = (F_{AD} \odot v_{BC}) \cdot \hat{v}_{BC} \tag{17}$$

$$F_t = F_{AD} - F_r \tag{18}$$

$$T_B = (P_A - P_B) \otimes F_t \tag{19}$$

$$T_C = -T_B \tag{20}$$

14.5.4.5. Electrostatic Interaction

An electrostatic force is calculated between atoms with a non-zero charge, as given by the standard coulomb function in Equation 21.

$$F_e = \left(\frac{-k \cdot q_i \cdot q_j}{(d_{ij})^2} \right) \cdot \hat{v}_{ij} \tag{21}$$

14.5.4.6. Hydrogen Bond Interaction

The hydrogen-bond function is a more complex function which uses two equilibrium distances. The first distance is between the hydrogen bond donor and acceptor atoms and the second equilibrium distance is between the hydrogen atom and the non-bonded electron pair of the acceptor atom, which is projected out from the center of the atom along a valence bond vector.

The function is piecewise continuous and consists of three regions: a lower distance region that is electrostatic, given by Equation 22, a middle region that is harmonic, given by Equation 23, and an upper distance region that is electrostatic, given by Equation 24. The force is then multiplied by a directional factor that reflects the orientation of the hydrogen bond.

$$F_{hl} = \frac{-k_e}{[d_{ij} - (d_{eq} - k_{wa})]^2} \tag{22}$$

$$F_{hm} = k_s \cdot (d_{ij} - d_{eq}) \tag{23}$$

$$F_{hu} = \frac{k_e}{[d_{ij} - (d_{eq} + k_{wa})]^2} \quad (24)$$

The lower and upper region electrostatic functions are derived by specifying a condition that the electrostatic force is equal to the harmonic force at some boundary point between the regions. The independent variables in the equations are the actual distance between the bond donor and acceptor atoms d_{ij}, the equilibrium hydrogen bond distance d_{eq}, the harmonic force spring constant k_s, and k_{wa}, which is actually a product of two constants that determine the width of the harmonic region and the slope of the electrostatic asymptote. Setting the two equations equal to each other at the boundary position allows for solution of the electrostatic scaling factor k_e.

The construction of the composite force function is illustrated in Figure 39, which shows a plot of force vs. distance for the three separate regions. The force for the lower region, f_L, is shown in red, the force for the middle region, f_M, is shown in blue, and the force for the upper region, f_U, is shown in green.

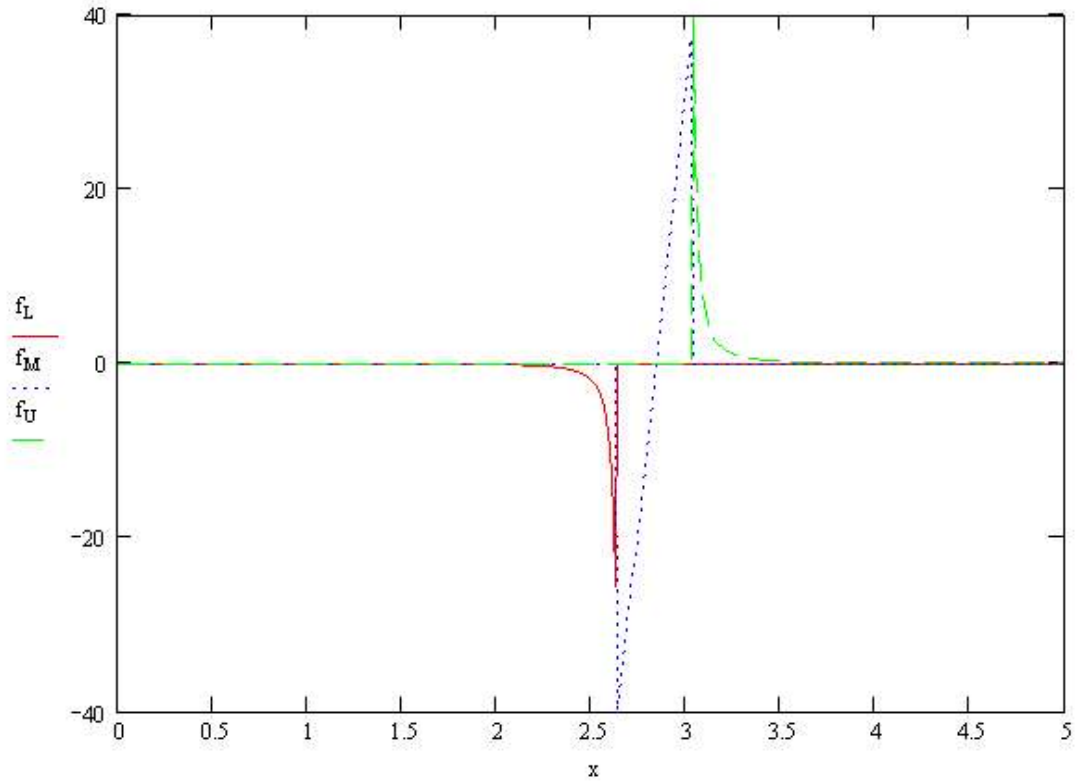

Fig. 39. Plot of the three component pieces of the hydrogen bond force function. The lower-region component is shown in red, the mid-region component is shown in blue and the upper-region component is shown in green. The x coordinate is the distance in Angstroms between the hydrogen bond donor and acceptor atoms.

The electrostatic and the harmonic regions reflect the two aspects of the hydrogen bond. The positive charge on the unshielded portion of the hydrogen atom interacts with the negative charge from the non-bonding electrons of the acceptor atom. This charge interaction is represented by the electrostatic regions of the force function, which have the desirable property of approaching zero with increasing distance between the two charges. The lower electrostatic function actually has a force value that decreases with decreasing absolute distance, but this corresponds to an increasing distance from the hydrogen bond equilibrium position. As the distance between the hydrogen bond donor and acceptor

decrease, the contribution from the hydrogen bonding interaction is overwhelmed by the very large steric forces that result from the overlap of inner-shell electrons. The middle region of the function produces a strong, linear change in force that reflects the covalent nature of the hydrogen bond. A plot of the combined regional force functions, along with the corresponding potential energy, is show in Figure 40. The potential energy plot is determined by piecewise integration of the force function.

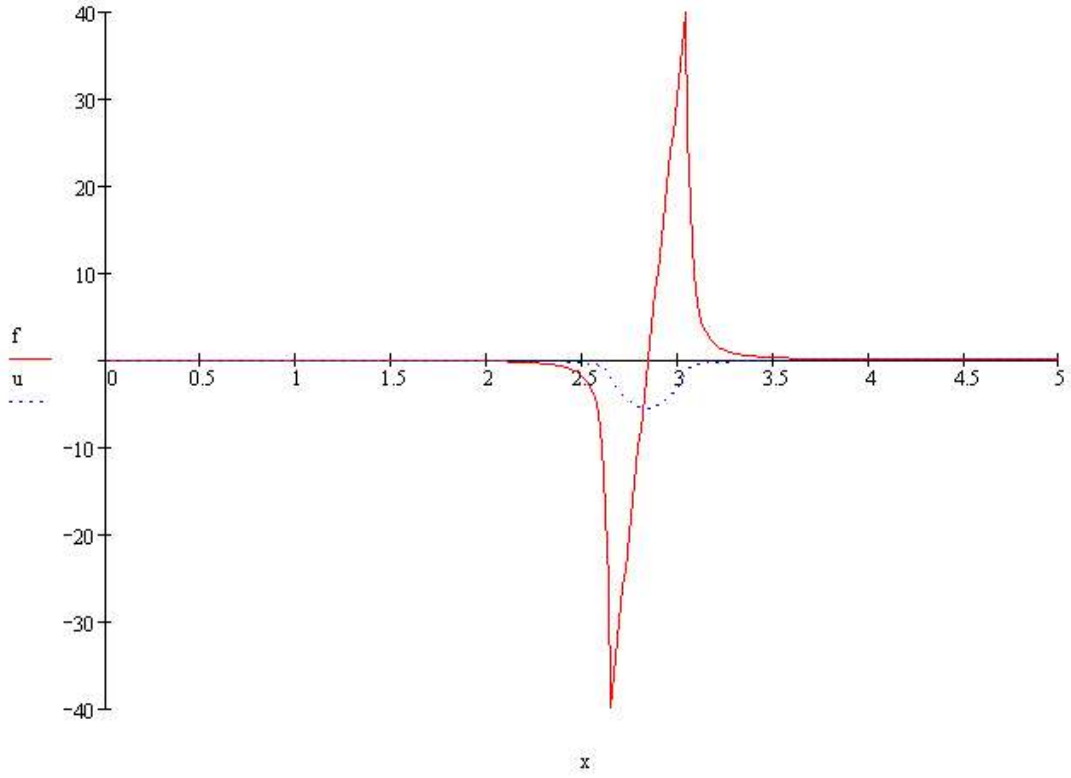

Fig. 40. Plot of the composite force (in red) and potential energy (in blue) for the hydrogen bond interaction. The x coordinate is the distance in Angstroms between the hydrogen bond donor and acceptor atoms.

The composite force function above was constructed to produce very strong linear hydrogen bonds, in order to put a greater emphasis on the mostly linear hydrogen bonds that are seen between bases that have Watson-Crick base-pairing in helices. However, many nucleic acid structures, especially RNA structures, have non-Watson-Crick base-

pairing in which hydrogen bonds can depart from a linear geometry. To address this greater variety of hydrogen bonds, an additional component was defined to supplement the hydrogen bond force function above. This second component defines a strictly electrostatic interaction between the hydrogen atom and the non-bonding electron pairs of acceptor atoms. The forcefield provides parameters that let the user adjust the relative strengths of these two hydrogen bond functions.

14.5.4.7. Base-Stacking Interaction

The aromatic base-stacking force is only applied for nucleic acid residues. It is currently implemented as a simple electrostatic force, calculated between centrally-located atoms in the aromatic ring, as shown in Equation 25.

$$F_r = \left(\frac{-k_r}{(d_{ij})^2} \right) \cdot \hat{v}_{ij} \tag{25}$$

14.5.5. Shadow Point Method for Covalent Bonding

In order to permit the use of strong driving forces and longer timesteps without excessive distortion of bond angles, a new method involving 'shadow points' was developed for the Ribosome Builder forcefield. This method differs from conventional molecular dynamics approaches, where bond angles are typically maintained by applying spring forces between 1-3 atoms.

The shadow point method relies upon the definition of a coordinate frame for each atom. This coordinate frame represents the geometrical orientation of the internal electronic structure of an atom. To define a covalent bond, two spring forces are applied between

each pair of covalently bonded atoms. One endpoint of each spring is connected to an atom and the other endpoint is connected to a shadow point. The shadow point represents the original bonded location of the first atom in the coordinate frame of the second atom. These springs constrain the relative movement of the coordinate frames of the two atoms to a rotation about their mutual bond vector.

Figure 41 illustrates the use of shadow points in covalent bonding. Coordinate frames are shown as x, y and z axes colored in red, green and blue, and located at the center of each atom. The shadow points represent the location of the bonded atoms when their coordinate frames are coincident. The divergence of the atoms from their shadow points has been exaggerated for illustration.

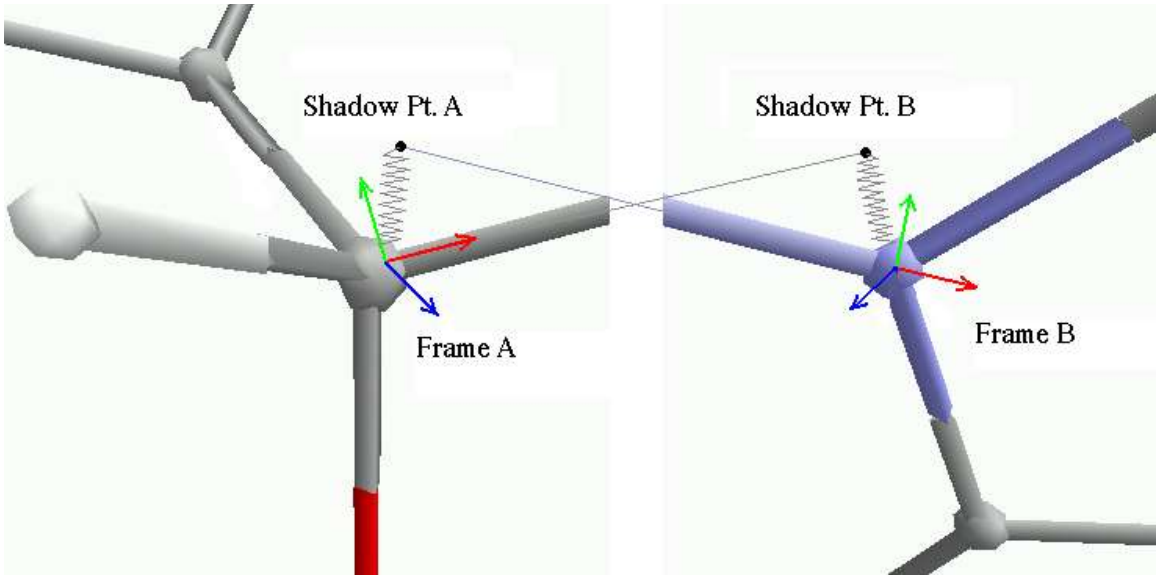

Fig. 41. Use of shadow points for covalent bonding. Coordinate frames are shown as x, y and z axes colored in red, green and blue, and located at the center of each atom. The shadow points represent the location of the bonded atoms when their coordinate frames are coincident. The divergence of the atoms from their shadow points has been exaggerated for illustration.

The shadow point constraints of multiply bonded atoms will then combine to maintain constant bond angles between the atoms. This effect becomes evident by considering a

single bond angle defined by three atoms, with atom 1 bonded to atom 2, which in turn is bonded to atom 3. As the coordinate frame of atom 2 rotates about its bond with atom1, the shadow point connection of atom 2 to atom 3 will also rotate about this same bond vector. This constrains atom 3 to rotate about the vector, maintaining the bond angle. This approach results in a rigidly bonded structure that is robust in response to strong external forces. The representation of the geometrical orientations of the electronic structure of atoms is also used in defining hydrogen bond interactions.

14.5.6. Forcefield Parameters

A large number of the parameters in the forcefield functions can be adjusted by the user, either through console commands or the scripting interface. A listing of these commands and their default values is shown in Table 1. In addition, there are a number of associated 'enabling' commands which can selectively enable or disable various component forces and other properties of the forcefield. The selective disabling or adjustment of forces can often be useful for identifying the principal force components involved in a particular molecular interaction.

Parameter	Description	Default Value
caElectroCalcEnabled	Enables electrostatic interactions between charged atoms	1
calcEnergyEnabled	Enables energy calculation in addition to force calculation	1
covalentFailDist	Sets covalent bonding fail distance (distance which bonded atoms can drift from bond limits)	0.2
covalentForceEnabled	Enables covalent bond force	1
covalentScaleForce	Sets scale factor for covalent bonds	100000
electroForceEnabled	Enables electrostatic force	1
electroScaleForce	Sets scale factor for electrostatic interactions	250
hbElectroCalcEnabled	Enables hydrogen-bonding electrostatic force	1
hbElectroScale	Sets the scaling factor for hydrogen bond electrostatic force	50
hbondForceEnabled	Enables hbond force	1
hbondScaleForce	Sets scale factor for hydrogen bond force	5
linFriction	Sets friction factor on linear movement	1
maxElectroDist	Sets maximum distance for electrostatic interactions (distance between charged atoms)	10
maxHbondDist	Sets maximum distance for hydrogen bonding interactions (distance between hydrogen and acceptor atom)	5
maxMove	Sets limit on maximum linear movement in a timestep	0.1
maxRingStackDist	Sets maximum distance for ring stacking interactions (distance between ring center-of-masses)	10
maxRot	Sets limit on maximum rotational movement in a timestep (in degrees)	1
maxStericDist	Sets maximum distance for steric interactions (distance between force object center-of-masses)	25
movementEnabled	Enables timestep movement	1
nbeProject	Sets projection of non-bonding electrons along atom radius, as a fraction of the vdw boundary	0.5
ringStackForceEnabled	Enables ring-stacking force	1
ringStackScaleForce	Sets scale factor for aromatic ring stacking interactions	100
rotFriction	Sets friction factor on rotational movement	0
sp2tTorsionScale	Sets the scaling factor for torque to maintain terminal sp2 moieties in bond plane	1000
stericAttractExp	Sets steric force attraction exponent	6
stericForceEnabled	Enables steric force	1
stericRepulseExp	Sets steric force repulsion exponent	12
torsionForceEnabled	Enables torsion force	1
vdwTorsionScale	Sets the scaling factor for van der waals radii used in torsion force calculation	0.8
volBounceEnabled	Enables bouncing off volume boundaries	0

Table 1. Forcefield parameters. These are the majority of forcefield parameters that can be modified at runtime.

14.5.7. Forcefield Movies

During a forcefield simulation, the movements of the atoms in the forcefield can be permanently captured for subsequently playback by enabling the recording the a forcefield 'movie'. A forcefield movie is a file that contains the spatial locations of all the atoms in the forcefield for a succession of timesteps, with a movie frame of data for each timestep. This implementation of a molecular dynamics trajectory is not the most efficient, and long movies for large dynamic models can grow to very large sizes. At one point, an all-atom simulation of the 30S subunit that ran for several weeks resulted in a movie file that exceeded the inherent 2 GB limit of a signed 32 bit integer, resulting in a program crash on an earlier version of the program. The bug was subsequently fixed, but a repeat of the simulation to verify it has not yet been done. Another problem with the existing movie file is that only atomic positional data are saved and any associated forcefield, graphical and annotation states must be reproduced independently.

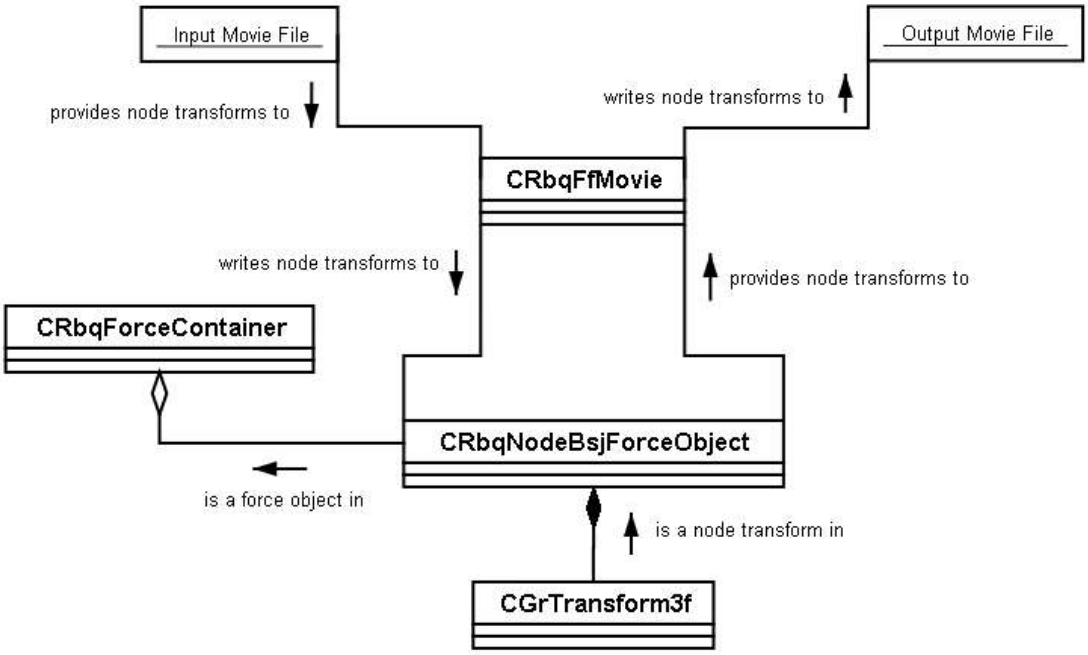

Fig. 42. Structural relationships for the CRbqFfMovie class.

The implementation of the forcefield movie is shown in Figure 42, with the CRbqFfMovie class acting as a simple relay for the coordinate data between the CRbqNodeBsjForceObject and the file.

14.5.7.1. Recording Options

The default method for recording movies is to turn recording on and then run the forcefield. Then, during each timestep, a call will be made to CRbqMovie to record the dynamic atom positions for that timestep. However, sometimes it is desirable to record movies when the forcefield is not running. One example is in schematic simulations that are doing geometrical interpolation of dynamic models. To allow the recording of the

conformational changes of models in such situations, there is an option for script-driven recording and playback of movie frames, independent of forcefield timesteps.

14.6. Scripting

Scripting is an important extension to the main application functionality through user-defined code that can be input at runtime. The script processing functions are encapsulated in the classes of the application-specific rbqLua library discussed above, but there are additional Lua components and interactions with the main application that are discussed here.

14.6.1. Lua Application Libraries

There are a number of higher-level functions written in Lua that are provided for the user in the form of Lua application libraries. A list of these libraries and their intended purposes is shown in Table 2. There are currently a total of 604 library functions that have been defined. These functions are used extensively in the definition of application-specific scripts created by the user. Examples of application-specific scripts include the definition of an SMD simulation or the analysis of torsion angles in a molecular model.

Name	Description	Module	Total Functions
Annotations	Functions to support annotations (such as text labels, lines, simple shapes, etc.) and schematic simulations	AnnUtil.lua	52
Camera	Functions to support camera operations (overlaps View functions)	CamUtil.lua	1
Chemical Drawing	Functions for displaying chemical representations (spacefill, covalent, surface, colors, etc.)	ChemDrawUtil.lua	28
Dihedral	Functions for dihedral and torsion angle operations	DihedralUtil.lua	27
File	File and directory functions	FileUtil.lua	30
Forcefield	Functions for controlling the forcefield	ForcefieldUtil.lua	24
Force Objects	Functions for operations on dynamic models	ForceObjectUtil.lua	91
Geometry	Functions for geometrical operations on static and dynamic models	GeomUtil.lua	30
Movies	Functions for forcefield movies	MovieUtil.lua	16
Pdb	Functions for data operations on static (PDB) models	PdbUtil.lua	47
Scripting	Functions for scripting and script objects	ScriptUtil.lua	63
Selection	Functions for selection operations	SelectUtil.lua	30
Sequence	Functions for biosequence-related operations	SeqUtil.lua	28
Tables	Utility functions for Lua tables (fundamental data type)	TableUtil.lua	15
Testing	Functions to support unit tests	TestUtil.lua	12
Vectors	Functions for vector and matrix operations	VectorUtil.lua	72
Viewing	Functions for controlling the view frame	ViewUtil.lua	29
Window	Functions for controlling the main application window and child windows	WinUtil.lua	9

Table 2. List of Lua scripting API function categories.

14.6.1. Lua Application Classes

In addition to the Lua library functions, there are a number of class objects that are pre-defined in the program. These class objects currently provide support for schematic simulations, which make heavier use object-oriented methods. The current Lua classes that have been defined include AnnUtil.Billboard, AnnUtil.Camera, AnnUtil.LineLabel,

and AnnUtil.Sim. Although the object-oriented approach in Lua scripting is useful and powerful, there is not as much support for type-checking and data encapsulation as there is with C++, so more complex utilizations of object-oriented Lua code in the application might lead to code-management problems.

14.6.4. Script Objects

Script objects are collections of functions and data that can be created by the user through a formal mechanism in the application. This allows the script object data to be persistent between invocations of a particular Lua script. There are also provisions for connecting script object functions to certain application events, along with a means for inspecting and modifying script object variables through the console interface.

14.7. Schematic Simulations

14.7.1. Definition, Purpose

A schematic simulation is a scripted sequence of events that is intended to represent one or more complex macromolecular processes to the user. The content of such a simulation draws upon a number of different capabilities from the main program, including loading and display of static and dynamic models, presentation of descriptive text and graphical annotations, coordinated movement of the three dimensional viewpoint, scripted translation and rotation of objects, and playback of pre-recorded movies of molecular dynamic simulations.

The term 'schematic' is used because this type of simulation is intended to fill a gap between a conventional two-dimensional description of events (as typically presented in a

journal article) and the completely detailed three-dimensional molecular dynamic simulation of some biological process. There are a number of benefits in this approach. First, the description of complex macromolecular structures and processes using conventional print media is inherently awkward and incomplete. A static two-dimensional image can only show a single view of a complex three-dimensional structure. Very often, it is not possible to adequately observe and identify partially occluded elements, especially in representations of large structures. In addition, dynamic movements cannot be explicitly represented. Therefore, one purpose of a schematic simulation is to act as a scientific description, similar to a conventional journal article, but using the medium of animated three-dimensional graphics.

A second purpose of the schematic simulation is to act as a starting point for the development of a fully complete model of some process. It is typical for a scientific model to evolve over time from an uncertain and incomplete state to a more detailed and well-supported one. In order to model a complex macromolecular process such as protein translation, a schematic simulation may employ low-resolution atomic models with uncertain positions, higher resolution atomic models with missing parts, a heterogeneous mix of molecules from different species, and a simplified set of movements that only symbolize the actual progression of atoms through time and space.

An important consequence of creating schematic simulations of macromolecular systems is that it motivates the definition of preliminary models for most of the principal components and interactions in a three dimensional space at an atomic level of detail. This results in an explicit, observable and detailed hypothesis. In turn, this facilitates critical evaluation of the hypothesis, which serves to identify missing or incorrect

components and states. An analogy can be made to a process that occurs in software development. Often, the design of a complex software project begins as a set of general specifications, which must then be implemented explicitly in source code. When this is happens, parts of the general specifications that are vague, incomplete or ambiguous will be glaringly revealed, because the actual implementation requires that such unspecified parts must be filled in before the work can proceed (Spolsky, 2004).

Finally, the components and interaction sequences that are created for a schematic simulation can often serve as reusable resources for the creation of additional simulations and the generation of alternative hypotheses. Over time, a powerful toolbox of predefined simulation components will grow. Collections of such components will typically be encapsulated into larger modules and will result in new building blocks that can be used to model processes at higher levels of abstraction.

14.7.2. Implementation History

Creating the capabilities needed to produce schematic simulations was not an obvious or simple task. No less than three distinct attempts were made during the course of the research project to implement a schematic simulation. Each attempt improved upon the mistakes and limitations of the previous ones. Because the design choices of the current implementation are best understood in that context, a brief summary of the two prior attempts is given below.

14.7.3. First Implementation, May 2003: Object-Oriented Lua Code

A schematic simulation draws upon different kinds of operations that are performed by the core application, such as loading models, setting graphical state of molecules,

constructing and displaying annotations. In order to do this in a flexible and customizable way, the development of a comprehensive scripting API was a necessary prerequisite. For this reason, the first schematic simulation was not created until the middle of 2003. The code to drive the simulation was created in the Lua scripting language. Also, because complex simulations are often a natural domain for the use of object-oriented design, a first attempt was simultaneously made to use of the object-oriented features of the Lua language.

The content of the implementation was to demonstrate the components and some of the steps involved in the initiation phase of translation on the ribosome. This included docking of mRNA, tRNA and initiation factors on the 30S subunit, and then docking of this complex with the 50S subunit. A means for doing geometrical interpolation of static models from one position and orientation to another was devised. A separate class was created to control the loading and setup of each component model in the simulation, with base classes created to handle some of the common operations. In addition, objects were created for controlling the camera movement and displaying text captions. An advantage of implementing the simulation in dedicated Lua code was that it enabled complete control and use of all operations that could be accessed through the scripting API.

The result was a fairly complex set of code modules that were all created in a common directory that was dedicated to this particular initiation simulation (still retained in the 'exp/sim/cycleSim' directory). This posed a significant problem for subsequent re-use of the code to create new, unrelated simulations. Also, the organization of code flow through a hierarchy of objects made debugging the simulation quite difficult, for two reasons. First, understanding the relationships between objects in a class hierarchy is inherently

more confusing than code flow in a flat, non-hierarchical design, unless the language provides strong support for encapsulation and static typing. This is not the case for the Lua language, or scripting languages in general. Second, the compiling and debugging tools used to create and run the code did not support source browsing, tracing and stepping through the Lua scripting code. As a result, the 'printf' style of debugging had to be used. Third, there was no good support for jumping between specific points in the simulation. The design was such that it always had be run from the beginning, up to a desired point. This made it tedious to investigate and debug specific subsections. Finally, one of the most severe limitations was that there was no good way to adjust the speed of the simulation when it was run of different computer systems which ran at a different speed from the initial development system.

This first schematic simulation was successfully created and demonstrated, but because of the limitations described above, and the large amount of work required to implement and debug it, additional simulations for other aspects of the translation process were not created with this initial approach.

14.7.4. Second Implementation, December 2003: SimPanel

The primary motivation in the second attempt at schematic simulations was to create a common framework that could be used to create multiple simulations. The formal term for a simulation at that time was HSIM (Hypothetical Sequences of Interpolated Movements). The framework was referred to as SimPanel (Simulation Control Panel, located in exp/dev/simpanel). At this time, a distinction was made between the simulation of a molecular model (Model Events) and the presentational aspects of the simulation, such as description and annotation (Presentational Events). The implementation

distinction between these two types of events was subsequently dropped in the final version for schematic simulations, although the conceptual distinctions remained.

The SimPanel implementation dropped the use of custom Lua code in favor of predefined script objects and functions. Only data was specified by the creator in order to define the simulation. This data-driven approach greatly simplified debugging and reusability of the code. Discrete 'steps' were defined for the simulations. These were called 'model events'. Breaking the activity up in this way made it possible to jump more easily between points in the simulation. This also greatly facilitated debugging and development of the simulation. Another important innovation was the addition of a scalar interpolation variable to specify the progress within an event. This enabled the speed of the simulation to adjusted in a simple way by changing the value of a global interpolation rate. Dedicated script objects were created to perform camera, text description and graphical operations. Quaternions were also added at this time to give a much improved and easier means for geometrically interpolating static models from one position and orientation to another.

The SimPanel scheme enabled the development of a new version of the translation initiation simulation, with much less time and effort than was required previously. A second simulation, representing the elongation phase of translation, was also created with this framework. Although the creation of this second simulation validated the reusability of the SimPanel, there was a new problem. In order to properly present the many complex aspects of elongation, very powerful control of graphical display and viewpoint was required. The difficulty of doing this revealed some significant limitations of the data-driven approach of the SimPanel. In order to make full use of the script API, each

operation would have had to be explicitly supported somehow in the SimPanel framework. It became clear that the ability to specify and execute custom scripting code, as was done in the first version, was still needed. Also, using only script objects for persistent data was awkward because of the need to interconvert all the data back and forth from the text representation that was used for maintaining script object state. Some way was needed that would combine the benefits of custom scripting of the first approach with the reusability of the SimPanel framework.

14.7.5. Third (Current) Implementation, September 2004: AnnUtil.Sim

The third implementation for creating schematic simulations combines the best aspects of the previous two approaches. It allows the use of custom code to define the behavior of the simulation. The framework for the code is organized in such a way that it is simple to jump to any position in the simulation sequence. This is done by creating an array of functions, where each function defines the behavior of the simulation in a discrete step. Each step receives processing time until it is completed, as determined by a scalar interpolation variable, which begins at 0 and is incremented by a default interpolation rate until it reaches the value of 1. When a step reaches completion, the next timer tick event will call a simulation driver object which in turn will call the function for the next step in the sequence. This repeats until all steps have completed. The entire simulation merely consists of a sequence of steps, where each step is a user-defined function. Figure 43 shows the principal components in a schematic simulation.

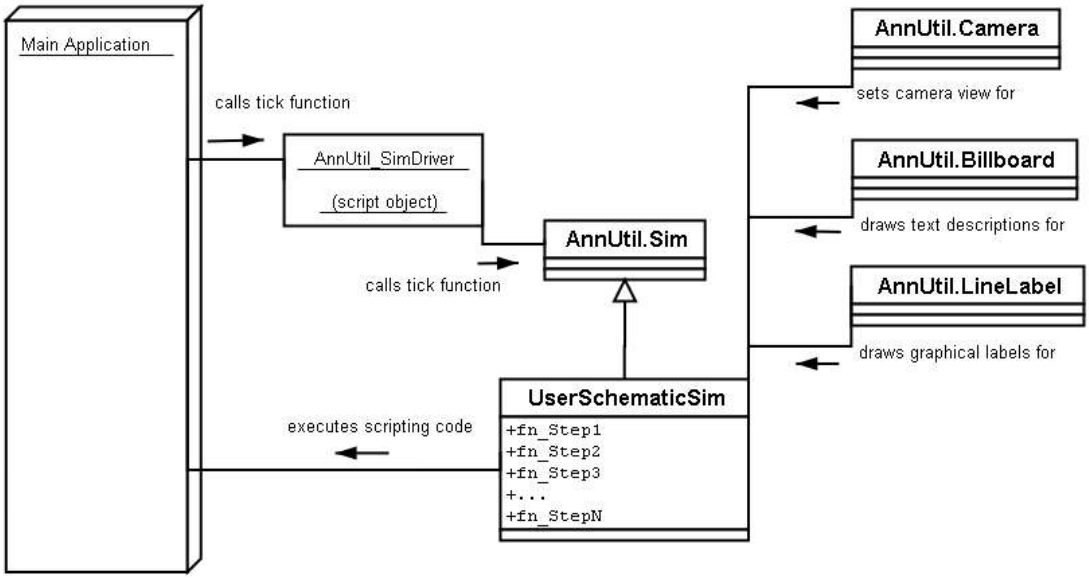

Fig. 43. Relationship between a user-defined schematic simulation module, supporting classes and the main application.

The class hierarchy is very simple, there is only a single base class, AnnUtil.Sim, from which the user-defined simulation object is derived. A number of classes to support the schematic simulation are also available, including AnnUtil.Camera, AnnUtil.Billboard, and AnnUtil.LineLabel. The camera class provides operations for doing smooth interpolations of camera position and direction. The billboard class manages the display of 2D orthographic text that is overlaid in the graphics window. The line-label class constructs arrows and associated text labels that are displayed in the 3D graphics window. These are used to point out specific features and objects to the user during the course of the simulation.

An important capability for the development and investigation of a schematic simulation is the ability to control the playback of the simulation. This control includes starting and stopping the simulation, jumping to an arbitrary location in the sequence of steps, and

single-stepping between and within steps. This run control capability allows the user to better examine specific aspects of interest within the simulation. A simple user interface for such a control panel can be implemented using an HTML browser, as is shown in Figure 44.

Fig. 44. Screenshot of an HTML browser that has been loaded with a set of controls for the execution of a schematic simulation in the main application window (not shown).

The layout of the page is done in a way that allows the browser window to be simultaneously displayed with the main application window. The hyperlinks within the browser define 'GET' requests that encode the file names of pre-defined Lua scripts, along with associated parameters. These predefined scripts are executed through the HTML browser interface, as discussed previously.

An example of an HTML-encoded request is the hyperlink for the 'goto step Next' text:

```
http://localhost:25546/?absScriptFile=exp%2fsim%2fcom%
2fgotoStep.lua&next
```

This causes the script 'exp/sim/com/gotoStep.lua' to be executed with the parameter string 'next'. The script looks for a current simulation object, retrieves the current step number, and then specifies that the simulation object should go to the next step number. In addition to the commands in the Annotation Step Control panel, a list below it also provides explicitly labeled steps, each of which can be clicked to set the current simulation step.

The first simulation script created using the current implementation was 'exp/sim/eln001', which is a complete schematic simulation of one cycle in the elongation phase of translation. The content of this simulation is described below in the application section.

The software implementation consists of a main script file called 'runSim.lua', along with a number of supporting files located in the directory. Within this main file, the array of step functions is defined. The simulation object itself is created by a call to the AnnUtil.Sim.new() function and a reference to it is returned and referred to using the local name 'sim'. This is a one-time operation that is performed with the script file is first executed.

Within each function, the progress of interpolation of the step is determined by calling the function sim:getInterp(). The interpolation value is used to set the progress of the geometrical interpolation of objects and the camera, and to coordinate the actions of sub-step events. The step function itself controls the progress of the step by explicitly incrementing the interpolation value through a call to sim:incInterp(). The documentation for AnnUtil.Sim provides additional details for defining and controlling the step activity. In addition to the array of step functions, the 'runSim.lua' script defines a number of special-purpose functions that provide common operations needed for the elongation simulation. Over time, some of these functions may migrate to the AnnUtil.Sim library if they retain commonly-used functionality. At the end of the script file is the main section, which processes command line parameters used in step control.

Part III: Applications

Chapter 15

Alignment of Ribosomal Subunits

15.1. Introduction

In this application, the issue of model alignment is discussed, along with the related need for a common geometrical reference frame. A standard for defining a reference frame for ribosomal models is proposed. This is followed by some background on aligning 3D structures, and a presentation of the Helix Alignment algorithm, which was created for the program. An explanation of how the algorithm works is done, along with an example alignment of two subunit structures.

When working with multiple models of ribosomal subunits simultaneously, it is frequently useful to geometrically align a pair of related models in the same 3D space. For example, 1GIX is a low resolution model of the 30S subunit. The model only defines the positions of the RNA and protein chain backbones. However, in conjunction with a second low resolution model, 1GIY, it represents a complete 70S ribosomal complex. By aligning two higher resolution models of the 30S and 50S subunits on this low resolution framework, a high-resolution model of the complete 70S complex can be approximated. Furthermore, the construction of such an alignment framework allows for placement of additional models of other ribosomal components, such as mRNA, tRNAs and elongation factors.

Typically, the coordinates assigned to PDB models are not created according to a common standard. For example, if the two SSU models 1FKA and 1J5E are loaded into the same 3D space, they will be at different positions and orientations. Although the absolute location of a particular PDB model could be chosen as the reference frame to which subsequent models would be aligned, this would be an arbitrary decision that would vary with the choice of the initial model. Therefore, in addition to performing alignment of models, there is a need to consistently locate the complete set of models with respect to the absolute frame of reference of the underlying coordinate space.

15.2. Defining a Standard Reference Frame

To address this problem, a standard geometrical frame of reference was defined for loading ribosomal models into the program. This standard frame was defined by loading the low resolution 70S model (1GIX/1GIY) of (Yusupov et al., 2001) and then aligning it to the absolute coordinate frame. This model was chosen because currently it is the most complete model of the ribosomal complex. It consists of both subunits, an mRNA fragment and three tRNAs. The alignment to the coordinate frame is done in the following way: 1) the center of mass of the entire 1GIX/1GIY model is positioned at the origin of the absolute coordinate frame. 2) the line joining the individual centers of mass of the two subunits is aligned with the X axis of the absolute coordinate frame. 3) the models are rotated about the x axis by 50 degrees in order to align the top of the subunits, as defined by the 'crown view', with the +y axis. These steps are encoded within the script 'load_1gix_and_1giy.lua'.

Once these reference components have been located in a standard way, then other ribosomal components from different PDB models can be placed in a standard location by aligning them with the reference components. There are a number of procedures for aligning models, as discussed below. Since the procedures can be somewhat complex, and vary from model to model, they are typically performed manually by the user. However, once a model has been aligned to a reference location, the homogeneous transform matrix of the model can be recorded and subsequently used to define a 'load' script. This load script can then be called to automatically place the model in the reference location. A number of such 'load_XXX_std.lua' scripts have been defined to load and place some of the models that are included with the program. A subset of the scripts are made available to the user in the 'Scripts', 'Models' menu in the program user interface.

15.3. Structure Alignment

Just as sequence alignment algorithms and software proved vital for managing the explosion of genomic data in the last decade, a similar need is developing for flexible and efficient geometrical algorithms and software to analyse and process 3D structural models. The number of molecular structural models continues to grow. In February, 2005, a milestone of 30,000 models was reached in the Protein Data Bank (PdbBeta site, 2005). Although the raw quantity and rate of production of structural data is far below that of primary sequence data, the 3D spatial domain provides additional complexities and challenges for theoretical development. Producing structural alignments of molecular models is one of the fundamental operations in this area.

There is no exact solution to the problem of 3D structure alignment, and so a variety of heuristic approaches have been developed (Shindyalov and Bourne, 1998). Furthermore, there can be multiple requirements that are involved, beyond solving the fundamental geometrical problem of superimposing structures. Some approaches are concerned with identifying and classifying the causes of structural divergence, such as evolutionary relationships. Other requirements include the ability to support multiple instead of pairwise alignments. Adequate performance of an algorithm is also a consideration for applications that must do alignments on large data sets such as the Protein Data Bank.

A number of successful implementations of structural alignment algorithms have been produced. Among the most common currently in use are DALI (Holm and Sander, 1994), VAST (Gibrat et al., 1996), and CE (Shindyalov and Bourne, 1998). All three of these programs are efficient enough to process the entire contents of the Protein Data Bank on a regular basis, and thus play important supporting roles in the useful access of its structural data. Given their performance and robust support, it may be desirable at some point in the future to integrate one or more of them, or related derivations, into the Ribosome Builder project. However, at the time the Ribosome Builder was being developed, a number of factors worked against such an incorporation. These factors included the increased algorithmic and implementation complexity of these tools as well as a general lack of availability of source code and supporting information. Instead, a much simpler method for performing structural alignments was developed for the program. This method, the Helix Alignment algorithm, has proved effective and adequate for satisfying the structural alignment needs in the research phase of the project.

15.4. The Helix Alignment Algorithm

The Helix Alignment Algorithm (HAA) is a method for aligning pairs of structural models of ribosomal subunits. The method relies upon the orientation of homologous helical elements of rRNA chains. Consequently, it is less general purpose than some of the more common alignment algorithms discussed above. However, it has a simple, straightforward implementation, and was a capability that evolved naturally from previous helical selection operations that were developed for the program.

15.4.1. Definition of the Helical Axis

The ideal geometry of a double-stranded helix is a straight linear cylinder. In practice however, the shapes of many helices depart from this uniform geometry with various bends, kinks and bulges. These variations result from asymmetries in the local sequence as well as from interactions with the surrounding environment. For the purposes of alignment, certain assumptions are made to unambiguously define an approximate helical direction in a simple and straightforward way. An imperfect approximation of the helical direction is not a problem in most cases because the direction is used for purely heuristic purposes to produce an optimal alignment. The ultimate measure of the alignment is done independently from such approximations, through the calculation of the RMS difference between the backbone atoms.

Therefore, the axis of a helix is defined by two points, A and B. Point A is the midpoint between the backbone atoms of residues R5b and R3e, where R5b is the beginning residue of the 5' side of the helix and R3e is the ending residue of the 3' side of the helix. Point B is the midpoint between the backbone atoms of residues R5e and R3b, where R5e

is the ending residue of the 5' side of the helix and R3b is the beginning residue of the 3' side of the helix.

15.4.2. Helix Pair Alignment

The actual alignment of the two models consists of three steps involving a translation and two rotations. The procedure is defined as follows: let the two PDB models be called model i and model j, where model j will be aligned to model i. For a given pair of helices H1 and H2 that are defined in both models, let the vectors of H1 and H2 in model i be named vH1i and vH2i, and let the vectors of H1 and H2 in model j be named vH1j and vH2j. In step 1, model j is translated so that the beginning endpoint of vH1j is aligned with the corresponding endpoint of vH1i (Figure 45A).

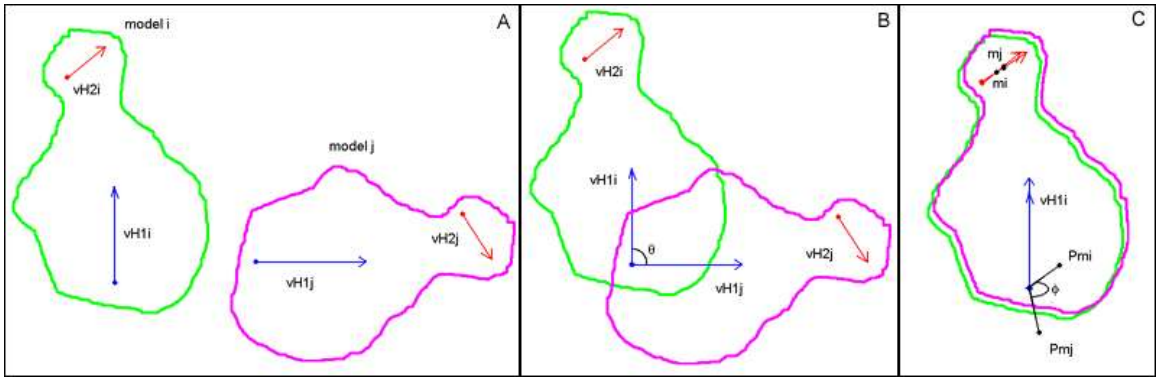

Fig. 45. Steps in the Helix Alignment Algorithm. **A.** Translation of model j so that endpoints of vH1i and vH1j coincide. vH1i is the vector of Helix 1 of model i, vH1j is the vector of Helix 1 of model j. **B.** Rotation of model j by angle θ to align vH1j to vH1i. **C.** Rotation of model j about vH1 by angle φ to align vPmj to vPmi. Pmi is the projection of midpoint mi of Helix 2 of model i into the plane of vH1i. Pmj is the projection of midpoint mj of Helix 2 of model j into the plane of vH1i.

In step 2, model j is rotated about an angle θ so that vH1j is aligned with vH1i (Figure 45B). In step 3, model j is rotated about an angle φ, where φ is the angle between vectors vPmi and vPmj, where these vectors are formed by projecting the midpoints of vH2i and vH2j into the plane defined by vH1i (Figure 45C). This results in the best possible

alignment of the second helix vectors while maintaining the alignment between the first helix vectors.

15.4.3. Finding the Best Pair Alignment

To find the pair of helices that will produce the best alignment between the two PDB models, each helix in model i is iterated. If the corresponding helix exists in model j, model j is transformed so that the helices are aligned as described in steps 1 and 2 above. Then, starting with model j in this intermediate transformation, each remaining helix in model i is iterated. If the corresponding helix exists in model j, then model j is transformed so that the second pair of helices are optimally aligned as described in step 3 above. After each alignment, the RMS difference is computed between the backbone atoms of all residues in the rRNA chains of the two models. The helix pair that produces the smallest RMSD value is then used for the actual alignment of the PDB models.

15.5. Example: Aligning Two Subunit Models

The actual implementation of the algorithm in the program is done with Lua scripts and draws upon a number of supporting operations and data. For each model in an alignment, the helices must be defined for the rRNA chain in that model. For a number of models that are included in the program distribution, the helix definitions have already been provided, in the form of .hlx files located in the seq subdirectory. For new models, the corresponding helix definition file can be generated by running a script called 'GenHelixFileForChain.lua'. This script uses the helix definitions for the canonical *E. coli* 16S and 23S rRNA chains.

The second requirement for helix alignment is a sequence alignment between the rRNA chains of the two models to be aligned. New sequence alignments can be created with the script 'CreateSeqMap.lua', which produces an .smap file in the seq subdirectory. The alignment is actually produced using a third-party library that has been incorporated into the program, the seqaln package (seqaln website, 2005). This provides an efficient alignment between two nucleic acid sequences based upon the dynamic programming algorithm described in (Waterman, 1995). It has proved especially effective for certain rRNA sequences that proved recalcitrant to other alignment programs such as CLUSTAL W (Thompson et al., 1994).

Once the necessary sequence map and helix definition files have been created, the alignment of the two subunit models in the program begins by running the script 'FindBestHelixAlign.lua'. This script, along with other alignment scripts, is available in the 'Script', 'Geometry', 'Align' menu. The script does an exhaustive search of all possible alignments, where each alignment is done from a pair of helices. The best alignment is determined by calculated the RMSD between the two models, using residue backbone atom positions. For example, if the two models 1GIX and 1J5E have been loaded into the program and the script is run, the program will iterate each defined helix in 1GIX. Even though there are 45 total possible helices, only 40 helices are actually iterated, because the same helix must be defined in both models or it will be skipped. For each iterated helix, the script then iterates all remaining helices shared by the two models and records the best RMSD alignment produced, along with the corresponding helix numbers of the pair that produced it. The results for all 40 iterations are then sorted in decreasing order and output to the user, as shown below:

```
Best Align 40: RMSD = 26.016 Helix1 =  38 Helix2 =  19
Best Align 39: RMSD = 17.805 Helix1 =  25 Helix2 =   2
Best Align 38: RMSD = 16.064 Helix1 =  33 Helix2 =   4
Best Align 37: RMSD = 14.956 Helix1 =  31 Helix2 =  6a
Best Align 36: RMSD = 14.527 Helix1 =   8 Helix2 = 33a
Best Align 35: RMSD = 14.240 Helix1 =  36 Helix2 =  42
Best Align 34: RMSD = 11.977 Helix1 = 23a Helix2 =  45
Best Align 33: RMSD = 11.692 Helix1 =  16 Helix2 =   1
Best Align 32: RMSD = 11.104 Helix1 =  30 Helix2 =   4
Best Align 31: RMSD = 9.942 Helix1 =  43 Helix2 =  15
Best Align 30: RMSD = 9.856 Helix1 =  6a Helix2 =   3
Best Align 29: RMSD = 9.241 Helix1 =   1 Helix2 =  44
Best Align 28: RMSD = 8.150 Helix1 =   2 Helix2 = 33a
Best Align 27: RMSD = 7.893 Helix1 =  42 Helix2 =  15
Best Align 26: RMSD = 7.677 Helix1 =  19 Helix2 =  35
Best Align 25: RMSD = 7.548 Helix1 =  20 Helix2 =  27
Best Align 24: RMSD = 6.684 Helix1 =  14 Helix2 =  44
Best Align 23: RMSD = 6.285 Helix1 =  35 Helix2 =  22
Best Align 22: RMSD = 6.211 Helix1 =  40 Helix2 =  15
Best Align 21: RMSD = 6.093 Helix1 = 33a Helix2 =  19
Best Align 20: RMSD = 5.978 Helix1 =  45 Helix2 = 33a
Best Align 19: RMSD = 5.700 Helix1 =  12 Helix2 =  42
Best Align 18: RMSD = 5.679 Helix1 =   5 Helix2 =  28
Best Align 17: RMSD = 5.501 Helix1 =  18 Helix2 =   2
Best Align 16: RMSD = 5.404 Helix1 =  28 Helix2 =   4
Best Align 15: RMSD = 5.301 Helix1 =  22 Helix2 =  37
Best Align 14: RMSD = 4.874 Helix1 =  15 Helix2 =  36
Best Align 13: RMSD = 4.855 Helix1 =  32 Helix2 =  42
Best Align 12: RMSD = 4.826 Helix1 =  44 Helix2 =  18
Best Align 11: RMSD = 4.807 Helix1 =  11 Helix2 =  18
Best Align 10: RMSD = 4.793 Helix1 =   3 Helix2 =  11
Best Align 9: RMSD = 4.732 Helix1 =  34 Helix2 =  6a
Best Align 8: RMSD = 4.635 Helix1 =  27 Helix2 =  42
Best Align 7: RMSD = 4.629 Helix1 =  23 Helix2 =   4
Best Align 6: RMSD = 4.468 Helix1 =  24 Helix2 =  35
Best Align 5: RMSD = 4.289 Helix1 =  21 Helix2 =  15
Best Align 4: RMSD = 4.176 Helix1 =   4 Helix2 =  37
Best Align 3: RMSD = 4.140 Helix1 =  13 Helix2 =  16
Best Align 2: RMSD = 4.108 Helix1 =  29 Helix2 =  6a
Best Align 1: RMSD = 4.026 Helix1 =  37 Helix2 =  18
```

Once the best pair of helices has been identified, the actual alignment is done with another script, 'AlignFromHelixPair.lua'. In this case, the resulting alignment is done using helices 37 and 18, to produce an RMSD value of 4.026 A between the backbone atoms of the two 16S rRNA chains. Because the rRNA defines the overall framework for

the subunit, the alignment generally produces good alignment of the embedded protein chains as well.

15.6. Conclusions

In this application, structural alignment of molecular models has been presented. The alignment of a pair of homologous models is useful for direct comparison of the variations in those models. In addition, structural alignment can be used to define an extended framework for the placement of multiple structural components, such as those of the ribosomal macromolecular complex. A standard for aligning the ribosomal framework to the underlying coordinate frame has also been proposed.

A number of algorithms and software programs in common use for structural alignment have been mentioned, along with the presentation of the Helix Alignment algorithm, which was developed for use in the Ribosome Builder program. The sequence alignment methods used in the implementation of this algorithm are also valuable for other operations, such as the mapping of residues from one species onto another. This is a frequent requirement in molecular modeling and analysis applications, where sequence positions are converted to a reference sequence that belongs to a corresponding reference species such as *E. coli*.

An example application of the algorithm has been shown, involving the alignment of two models of the 30S subunit, with pdb codes 1GIX and 1J5E. The algorithm has also been applied to the alignment of high resolution models of the small and large subunits on a lower resolution frame of the 70S complex. One application where this has been done was an investigation of the bridging contacts between the two subunits that form upon

their association (Hennelly et al., 2005). Although this alignment algorithm is special purpose in nature, and is likely to be supplemented by more general structural alignment tools in the future, it has served a vital and successful role in enabling the construction of multi-component macromolecular models in the present phase of the program's development.

Chapter 16

Folding a Tetraloop

16.1. Motivation

Tetraloops are important structural motifs in RNA and are present in numerous locations in ribosomal RNA (Woese et al., 1990). A tetraloop is a single-stranded sequence of 4 unpaired nucleotides that loop out from one end of a double-stranded helical segment (Moore, 1999). Because it is a stable, common and important RNA fold, a GNRA tetraloop was chosen as a reference structure to test the SMD folding capabilities of the Ribosome Builder program. A particular GCAA tetraloop motif has been previously investigated in ensemble molecular dynamics studies (Sorin et al., 2002), (Sorin et al., 2003), (Sorin et al., 2005). The most recent of these studies has just produced a successful simulation of complete folding of the tetraloop from an unfolded state through the use of a massively distributed computing infrastructure. However, at the time when folding simulations for this dissertation were first undertaken, a successful atomistic explicit solvent MD simulation of complete tetraloop folding had not yet been accomplished. The target tetraloop motif chosen for the Ribosome Builder SMD simulations was obtained from the terminal residues of helix 41 of the 30S subunit in the crystal structure of PDB model 1J5E (Wimberly et al., 2000). A secondary structure diagram of the helix is shown in Figure 46 and the tertiary structure from the PDB model is shown on the right in dark green in Figure 47A. A subset of 12 residues from the helix were actually used,

starting with C1262 and ending with G1273. These residues form the 4 base-pairs at the end of the helix stem plus the 4 residues in the tetraloop, GCAA. Note that C1262 and G1273 are not shown as based-paired in the secondary structure diagram (CRW website, 2005), but they are in fact paired in the PDB model.

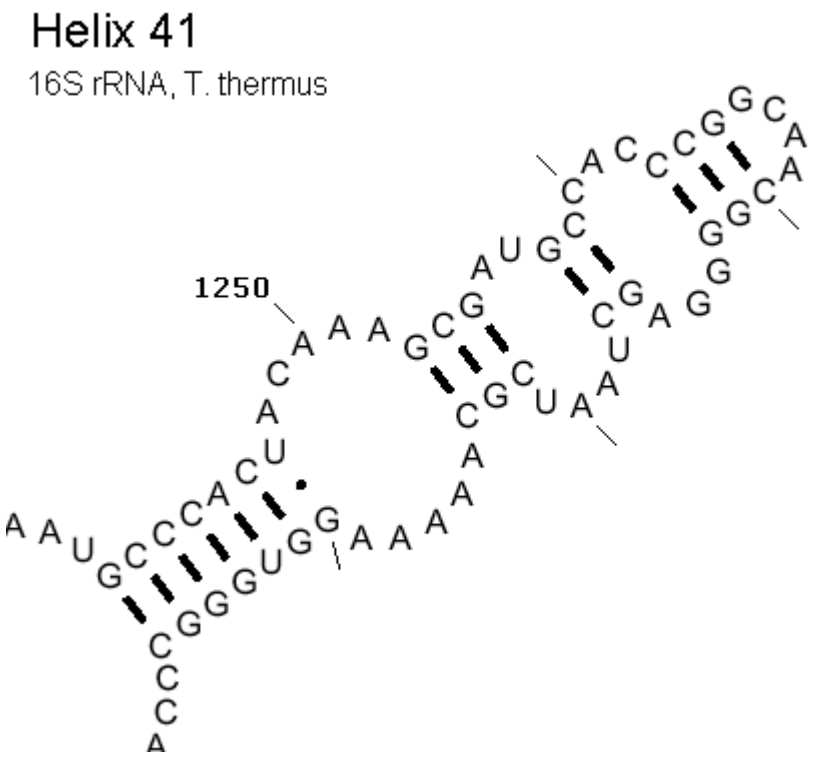

Fig. 46. Secondary structure diagram of Helix 41 of 16S rRNA from *Thermus thermophilus*, PDB code 1J5E. The numbering shown is the actual sequence numbering, not *E. coli* numbering . Diagram is from (CRW_website, 2005).

The next step was to define a suitable starting structure for the simulation. The requirements were that it should be a structure of known stability, but also one that was significantly different from the target structure so that significant conformational change would be required to achieve the target fold. At the same time, a starting structure that mandated an excessively complex conformational change would not be desirable for an early test case.

An analysis of the backbone torsion angles of the folded tetraloop structure was done using the dihedral functions in the scripting API of the program. The torsion angles were classified into the rotamer scheme of (Murray et al., 2003). This scheme is a proposal for classifying the conformations of RNA backbones into a set of patterns called 'rotamers'. The notion is similar to the concept of protein rotamers, which are clusters of side chain conformations that are observed in the distribution of protein structures (Lovell et al., 2000). These clusters are presumed to occur because they are associated with lower conformational energies. The classification of RNA backbone conformations is more problematic than that of proteins, because of the larger number of torsion angles per residue, as can be seen in Figure 3. A number of other attempts at classification have been made including (Duarte and Pyle, 1998), (Murthy et al., 1999) and (Hershkovitz et al., 2003). The scheme of Murray et al. is notable for its concise and explicit formulation, which lends itself to independent computational implementations. Using the torsion angle frequency analysis tools of the Ribosome Builder program, classifications of chains from ribosomal models were made according the Murray rotamer codes. Although many residue suites fell within the 42 Murray codes, there were significant numbers of residues that could not be classified in this way. This is an indication that the Murray system may need subsequent revision to account for additional conformational structures. In spite of this, their code is a useful starting point for simplifying the large conformational complexity of the RNA backbone.

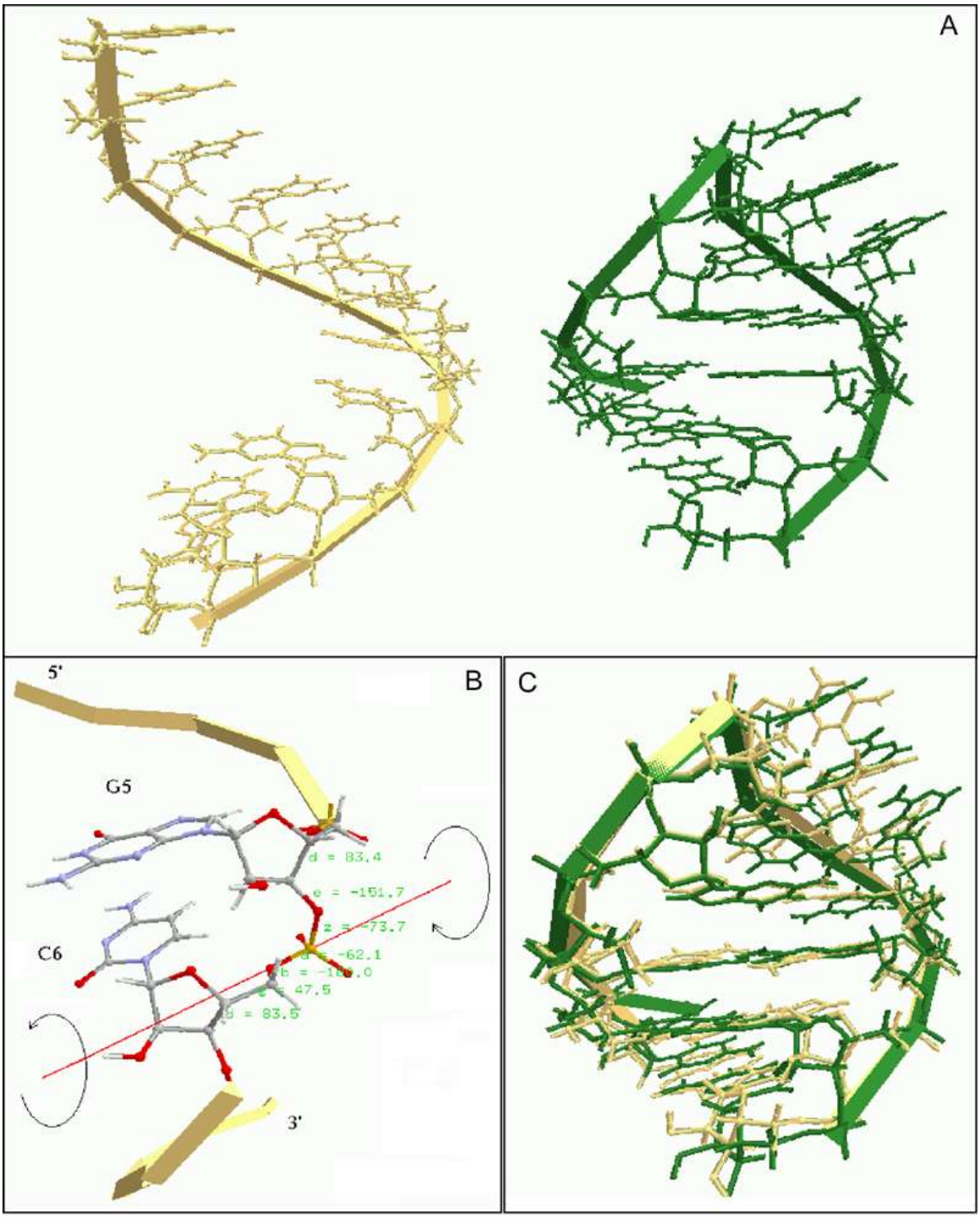

Fig. 47. Tetraloop folding. A. Unfolded oligomer in tan on left, target structure in dark green. B. Directions of applied torque and reaction torque at torsion angle α between residues G5 and C6 of unfolded structure. C. Alignment of folded structure and target structure.

The rotamer analysis of the folded structure showed that all of the residue suite conformers were in the A-form helix conformer except for the single suite of the first two residues of the tetraloop (G1266 - C1267). This indicated that a structure starting in A-form helix would be suitable for meeting the requirement of significant, but not too complex, conformational change. A structure with the target sequence was then generated in the helical conformation using the Nucleic Acid Builder software program (Macke et al., 2004). This starting structure, referred to as the 'folding oligo', is shown on the left in tan color in Figure 47A.

16.2. Defining the Steering Forces

To drive the oligo to the folded state, an SMD script was created to apply angular springs around the bonds in the backbone atoms. An angular spring, also called a 'torsion spring', applies a torque on an atom and an oppositely-directed reaction torque on its bonded partner atom, with a magnitude proportional to the angle difference between the current torsion angle and the target torsion angle for that torsion bond. An indication of the applied torques that will result is shown at the P-O5' bond between G5 and C6 in Figure 47B. These two residues in the folding oligo correspond to G1266 and C1267 in the target oligo, and this torsion angle is the one that differs most between the two structures, with an initial value of -145.7 °. A total of 68 torsion springs were created for all of the torsion angles in the folding oligo. For each torsion spring, a force constant variable defines the stiffness of the spring and a limit variable defines the maximum torque that will be applied. The clamping of a torque to a maximum value is necessary to prevent

excessive distortions of the model when the difference in folding and target torsion angles is very large.

The primary motion of the oligo is a twisting of the two halves on either side of the C6 α torsion angle described above. Initial simulations revealed that the driving torque for this torsion angle had to be defined differently from the other torques. By default, the torsion angle script applies torques in the direction that will most directly minimize the difference in torsion angles. However, for the angle above, trying to torque the bond in the default direction resulted in a steric clash between adjacent bases, and the folding became stuck. This can be visualized by observing the potential counter-clockwise rotation of the base ring of G5 into that of C6 in Figure 47B. Presumably this directionality reflects the way the oligo might fold in reality, although it is possible to envision a more complicated set of coordinated rotations of adjacent residues that would enable a clockwise folding of the two ends.

The imposition of a directional constraint on a particular torsion bond required more than just a simple sign change of the driving torque. If the driving torque direction was strictly held constant, then when the clockwise rotation of the bond resulted in a coincidence of the folding and target torsion angles, any additional slight increase of the folding angle in a clockwise direction would result in another complete cycle of rotation, or at least an attempt to do so. In order to avoid this undesired effect, the user-specified directional constraint had to be applied only when a certain threshold of divergence occurred between the folding and target angles.

As this was the first significant test of steered molecular dynamics in the program, quite a few simulations were done to determine the optimal conditions for folding. At the same

time, a process of identifying and fixing various bugs in the software was taking place. Fairly early on, the simulations were able to successfully rotate the oligo about the G5-C6 bond so that the two ends of the oligo were in position to form base pairs in the stem. At this point however, some sampling issues came up. With the default hydrogen bond parameters, a few initial strong and stable hydrogen bonds formed between bases in the stem that were not necessarily those of the final target configuration. Because of the stability of the pairs, the simulation remained trapped in these local energy minima. A number of things were tried to increase the sampling of contacts during the simulation, including the addition of ions, random thermal inputs, changes in frictional parameters, and very long simulation times. In the end, the best results were obtained by reducing the strength of the hydrogen bond forces to a threshold level, allowing tentative base pairs to form and break. With this approach, the most successful simulation achieved a close match to the target structure by forming base pairs between all 4 bases in the stem at timestep 2188.

During the simulation, a large degree of flexibility is evident. One issue that may or may not be relevant to the overall success of the simulation has to do with rotations of terminal groups on atoms with sp2 valence structure. At the time this simulation was done, the existing version of the forcefield did not impose constraints on the rotation of sp2 covalent bonds. For example, the two terminal hydrogens of the N6 exocyclic amine of adenine can be observed to freely rotate outside the plane of the base ring. Likewise, the lone-pair electrons on O2 of cytosine can be seen to undergo similar rotations. A later version of the forcefield added a constraint on these kinds of bonds so that they tend to remain in a planar configuration. The addition of these constraints resulted in improved

base-pairing in helical structures during later simulations of larger structures, but the impact of the newer constraints on these earlier tetraloop folding experiments remains to be fully investigated.

16.3. Results

A movie of the successful simulation mentioned above was recorded and was included as a demo in the program distribution. The superposition of the folding structure at timestep 2188 was manually superimposed on the target structure and is shown in Figure 47C. There is fairly good overlap of the backbones of the two fragments and most of the bases are closely aligned in position. A divergence of the base ring of C6 from the corresponding target base C1267 is quite noticeable however. This may be due to an inadequate base-stacking function in the forcefield, as well as some other possible deficiencies. These include the lack of an explicit solvent as well as counter-ions.
The ability of the SMD driving script to successfully achieve the target conformation can be assessed from the torsion angle data in Table 3. The data in the table were generated and formatted using Lua and Perl (Wall et al., 1996) scripts that were created for torsion angle analysis. The table shows the 6 backbone torsion angles for each of the residues in the folding and target structures, as well as the difference between them. Most of the angles have a difference of less than 5 °. The largest difference is at the C6 α torsion. These differences most likely reflect the dynamic conditions the oligo was experiencing during folding.

Res. Num.	Res. Name	Alpha	Beta	Gamma	Delta	Epsilon	Zeta
1	folding.C1	-	-	51.144	83.596	-166.374	-60.775
	target.C1262	-	-	53.632	86.737	-164.258	-60.534
	Δ	-	-	-2.488	-3.141	-2.116	-0.241
2	folding.C2	-79.025	-172.314	54.743	79.756	-153.248	-74.471
	target.C1263	-78.546	-172.139	48.390	82.436	-156.983	-72.038
	Δ	-0.479	-0.175	6.353	-2.680	3.735	-2.433
3	folding.C3	-71.943	176.628	65.727	81.111	-148.444	-78.466
	target.C1264	-67.549	172.548	60.505	83.304	-153.303	-73.574
	Δ	-4.394	4.080	5.222	-2.193	4.859	-4.892
4	folding.G4	-81.747	-173.447	54.441	87.572	-143.739	-67.359
	target.G1265	-78.764	-173.051	47.965	80.856	-144.847	-65.923
	Δ	-2.983	-0.396	6.476	6.716	1.108	-1.436
5	folding.G5	-67.134	175.630	49.883	84.549	-132.218	-65.645
	target.G1266	-70.570	179.341	55.344	81.821	-129.882	-66.207
	Δ	3.436	-3.711	-5.461	2.728	-2.336	0.562
6	folding.C6	175.821	148.395	48.053	83.211	-136.811	-68.576
	target.C1267	152.171	158.174	54.566	89.400	-134.691	-65.238
	Δ	23.650	-9.779	-6.513	-6.189	-2.120	-3.338
7	folding.A7	-67.354	164.474	69.626	85.970	-150.771	-80.423
	target.A1268	-61.743	164.799	59.385	87.857	-154.580	-81.802
	Δ	-5.611	-0.325	10.241	-1.887	3.809	1.379
8	folding.A8	-84.559	169.876	61.409	87.770	-122.922	-47.268
	target.A1269	-75.111	166.704	60.562	89.926	-120.169	-49.958
	Δ	-9.448	3.172	0.847	-2.156	-2.753	2.690
9	folding.C9	-58.653	168.983	70.901	80.864	-154.639	-72.153
	target.C1270	-53.429	150.996	68.961	81.936	-156.489	-77.294
	Δ	-5.224	17.987	1.940	-1.072	1.850	5.141
10	folding.G10	-65.313	165.310	58.916	83.095	-159.244	-67.532
	target.G1271	-62.004	171.573	56.993	81.030	-152.147	-65.297
	Δ	-3.309	-6.263	1.923	2.065	-7.097	-2.235
11	folding.G11	-70.280	-169.804	48.270	87.180	-160.275	-56.495
	target.G1272	-72.374	-173.329	52.909	82.289	-157.951	-57.553
	Δ	2.094	3.525	-4.639	4.891	-2.324	1.058
12	folding.G12	-56.114	177.832	54.138	82.882	-	-
	target.G1273	-72.566	175.546	58.924	78.740	-	-
	Δ	16.452	2.286	-4.786	4.142	-	-

Table 3. Comparison of tetraloop final folding and target torsion angles. Differences in angles that are between 1 and 10 degrees are highlighted in orange. Differences between 10 and 60 degrees are highlighted in red.

16.4. Conclusions

Although this simulation was done with rather stringent artificial constraints to produce a fold to a known target conformation, it was able to produce a chemically plausible trajectory in which bond lengths, angles and steric exclusion between atoms were maintained throughout the folding. In this respect, it shares similarities with other efforts to use high-level constraints for long time scale simulations such as the folding of the anti-codon stem loop of tRNA (Harvey et al., 2003). More accurate MD simulations of tetraloop dynamics would be preferred for determining the actual stability of a target fold, but in the past, they have been generally confined to simulating smaller fluctuations around a particular conformation (Sorin et al., 2002). Very recently, massively-parallel ensemble molecular dynamics have been able to reproduce this long-term conformational change, with considerably greater precision in evaluation of the energy potential. However, this has only been made possible by the exploitation of huge computational resources, equivalent to 150,000 CPUs (Sorin et al., 2005).

Further refinement of the forcefield and folding conditions can certainly be done, but the tetraloop simulation fulfilled its primary goal as an initial test application to validate the Steerable Molecular Dynamics capabilities of the Ribosome Builder program. The simultaneous application of multiple externally-applied torques proved able to induce significant changes in the nucleic acid backbone and achieve a desired target

conformation. Furthermore, the flexibility of the SMD approach was demonstrated in several ways. It was possible to dynamically adjust torque constants and thresholds while the simulation was taking place. The scripting approach also made it possible to implement complex driving functions, such as user-specified directionality of external torques and conditional thresholds for their application.

The SMD capabilities demonstrated by the tetraloop simulation made it possible to proceed to more complicated simulations such as those investigating conformational changes in mRNA during ribosomal translation. The approach of externally-applied torques on backbone torsion angles may also prove useful for discovering more general principles of folding and dynamic transitions of nucleic acids. The rotameric definition of RNA conformation (Murray et al., 2003), in conjunction with externally-applied torques, may prove useful for investigating the existence of characteristic coordinated transitions of adjacent rotamers along an RNA backbone. This would be done by defining a set of torques to induce the transition of an individual residue (or residue suite) from one rotameric conformation to another. Then, the ability or inability to produce a coordinated set of transitions for a sequence of adjacent residues may serve to identify the subsets of coordinated transitions that are possible.

These coordinated transitions, if they exist, might be termed 'convertamers'. Just as RNA rotamers are characterized as distinct subsets of static conformations out of all possible backbone conformations of a residue (or residue suite), convertamers might be considered as distinct subsets of dynamic conformational transitions of a short sequence of adjacent residues relative to all possible conformational transitions of those residues. The characterization of convertamers might help to better identify and understand 'switching'

activities of RNA such as the proposed conformational switch involved in decoding of mRNA (Lodmell and Dahlberg, 1997). This notion of the capability of residues to undergo transition between two distinct conformations has also been associated with the term 'multispecificity' in the context of the switching function of G-proteins (Biou and Cherfils, 2004).

Of course, the totality of constraints on conformational transitions of RNA will include much more than the backbone torsion angles. Steric interactions, base-stacking and hydrogen-bonding are also part of the process. However, as a fundamental determinant of position for the entire nucleic residue, the backbone torsion angles form a natural starting point for the possible characterization of RNA convertamers.

Chapter 17

Wrapping mRNA

17.1. Introduction

A prerequisite for an adequate simulation of normal translation is a model of an mRNA of sufficient length that it spans the width of the ribosome. During the elongation phase of translation, the two active codons of an mRNA occupy the A and P sites on the 30S subunit and serve as the focus of activity during the decoding step. These two central codons consist of 6 residues, but an mRNA of approximately 30 residues is needed to extend from one side of the 30S subunit to the other (Yusupova et al., 2001).

In contrast to the great deal that is known about tRNA, there is much less detailed structural data on mRNA interactions with the ribosome (Culver, 2001). Some low-resolution cryoEM structures have given indications of a possible channel for mRNA that forms on closure of the head and shoulder elements of the 30S subunit (Lata et al., 1996). The first atomic resolution crystal structures of the 30S subunit show the placement of an rRNA fragment in the P-site. This fragment, which is most likely the 3' end of the 16S rRNA, could be acting as kind of mRNA mimic (Carter et al., 2000). Other non-structural data have also contributed to the understanding of mRNA binding. One classic feature is the anti-Shine-Dalgarno sequence at the 3' end of the 16S rRNA, which interacts with Shine-Dalgarno sequences that are commonly found at short distances upstream from the initiation codon on the mRNA (Gualerzi and Pon, 1990).

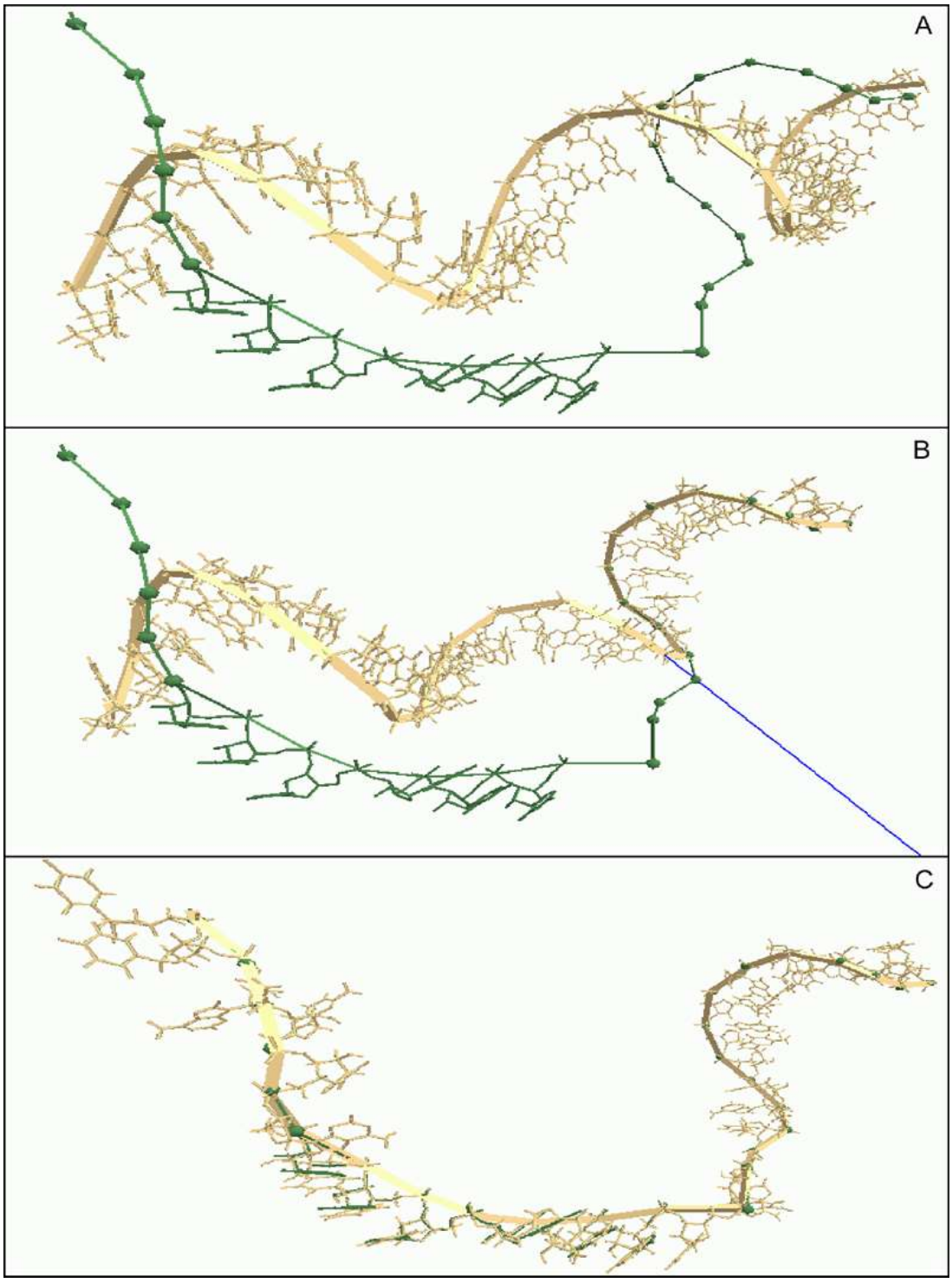

Fig. 48. Wrapping an mRNA around a target path. A. Unfolded mRNA in A-form helix colored in tan, target structure in dark green. B. Partially wrapped mRNA, with blue line showing external force acting on currently aligning residue. C. Final conformation of mRNA wrapped around target structure.

An incomplete crystal structure model of an mRNA docked to the 30S subunit has been produced (Yusupova et al., 2001). This model was obtained from x-ray diffraction data on 70S ribosomal complexes with bound tRNAs, both in the presence and absence of an mRNA fragment. This allowed for an approximate determination of the mRNA structure from 7 A Fourier difference maps. The resulting model of the mRNA (PDB code 1JGO) is 27 nucleotides in length, shown as the dark green structure in Figure 48A. However, the model only defines the 6 center nucleotides in full atomic detail. The remaining residues are represented by the positions of backbone phosphorus atoms. This represents an extension of the previous model of the 6 central residues of mRNA in the crystal structure of the 70S ribosomal complex at 5 A resolution (Yusupov et al., 2001).

17.2. Procedure

This partial model motivated the creation of a second SMD application for the Ribosome Builder program. In this application, an all-atom model of a 27 nt RNA oligomer was generated in standard A-form helix conformation by Nucleic Acid Builder, and is shown as the tan structure in Figure 48A. An SMD script was then created to drive the oligo to the target conformation of the docked mRNA in the Yusupova crystal structure. In contrast to the torsion springs used for tetraloop folding, the mRNA folding script defined linear spring forces that were applied between the backbone phosphorus atoms of the folding oligo and the corresponding atoms in the target mRNA structure. Another difference in the mRNA alignment was to use a strategy of sequential application of driving forces. It was felt that trying to align all of the backbone atoms simultaneously

might cause the folding oligo to become trapped in an intermediate structure. In addition, the sequential approach simplified the software debugging process, because flaws in the driving code were more likely to be identified and fixed if they were applied to a single residue at a time. A further simplification was done for this initial mRNA docking by skipping the simulation of interactions between the folding mRNA and the surrounding residues of the 30S subunit.

The sequence of the mRNA in the Yusupova model is "GGCAA GGAGG UAAAA UUUUU UAAAA AA". However, the actual sequence used for the folding oligo was "AUAAG GAGGU AUACU AUGUU UACGA UU". This sequence contains an initiation codon and was obtained from an mRNA used in experiments for investigating the formation of initiation complexes (Scott Hennelly, personal communication). The Yusupova model was loaded as a static model to act a source for the target backbone atom positions. The first residue of the A-site codon is defined as position +1. In the target model, this is residue U19. Therefore, there are 18 residues upstream from position +1 (G1 - U18), and 8 residues downstream (U20 - A17).

The simulation began by applying a force between the phosphorus atom of the first residue of the folding oligo and the corresponding atom on the target residue G1. Like the torsion spring script in tetraloop folding, the spring force function used a spring constant value to determine the magnitude of the pulling force and also a maximum threshold value to prevent excessive distortions of the folding model. In addition, a second threshold distance was defined. This threshold value acted as a condition for indicating that a residue was close enough to its target. When this condition was met, the next

residue in the sequence became enabled for the application of its aligning force. As the alignment proceeded, residues further down the oligo would begin to be pulled toward their respective targets. While this was occurring however, the aligning forces for the preceding residues in the chain were still being applied, so that the previous portion of the oligo would remain in alignment. Optionally, graphical indicators of applied forces could be displayed, drawn from their source atom towards the target, and with a length that is proportional to the magnitude of the applied force. An example force indicator is shown as a blue line in Figure 48B.

In addition to the external forces, the interactions between the residues were simulated by the forcefield in order to maintain the proper steric and covalent bond distances and angles during the alignment. About two hours of real-time computation on a typical desktop machine were required to produce a successful alignment of the backbone atoms to the target locations, resulting in an all-atom model of an mRNA in the docked location (Figure 48C). A movie of the simulation was recorded and is included in the program distribution.

17.3. Results

As shown in the recorded movie of the wrapping process, the mRNA aligns to the target path fairly smoothly in the beginning part of the simulation. The alignment took place in the context of a small amount of frictional forces applied by the forcefield, which was done to enhance the stability of the folding. This is evident in the relative inertia of the terminal portion of the chain as the active portion is aligned. When the alignment reaches target residue U16, which is at position -3 from the start position, a more abrupt type of

motion is observed. This is because this portion of the target segment consists of fully-resolved residues, so that all the atoms of the backbone were available for alignment, as opposed to the single phosphorus atoms of the unresolved residues. As the alignment proceeds past the codon in the A-site, the inertia of the trailing residues becomes more pronounced, possibly to due a natural 'bunching up' of the residues in the absence of external influences. This may indicate the need for a more realistic simulation that involves explicit solvent molecules and counter ions. In accordance with the threshold distance condition of the simulation, the completed alignment resulted in the docking of the backbone atoms of the mRNA to within 0.2 A of the target positions.

17.4. Conclusions

This application has involved the use of external spring forces to pull a 27 residue model of mRNA from an A-form helical conformation to one that traces a path through the neck of the 30S subunit. The simulation produced extensive conformational changes in a significant length of mRNA, with the result that a desired set of target residues positions were reached. This has served to validate an additional component of the steerable molecular dynamics capabilities of the program.

In this initial simulation of mRNA, interaction with the surrounding rRNA and proteins of the 30S subunit was not done, but the success of the approach opens the way to simulations of more realistic trajectories of mRNA docking. A number of steps in the translational process, including the Shine-Dalgarno interaction and translocation, will require similar productions of large conformational change of extended mRNA molecules. One particularly interesting area might involve the interactions of the 3' end of

the mRNA with ribosomal proteins S3 and S4, which have recently been implicated in a possible helicase activity of the ribosome (Takyar et al., 2005). The more immediate result from this simulation is the acquisition of a full atomic resolution model of an mRNA that is docked on the 30S subunit. This model was then used as a component in a subsequent schematic simulation of the larger process of the elongation cycle of the ribosome.

Chapter 18

Schematic Simulation of the Elongation Cycle

18.1. Introduction

In contrast to the view of static stereochemistry of components at fixed sites, a dynamic concept of translation is focused on the changes that occur, including the transitions between states (Woese, 2001). Translation of mRNA into protein on the ribosome can be divided into three distinct phases: initiation, elongation and termination. A complete model of the elongation phase remains to be developed, but there are now a sufficient number of structural models of components and predicted events for the decoding, peptide-transfer and translocation substeps to permit the construction of a schematic model of the entire elongation sequence at an atomic level of detail. A preliminary version of such a model is presented here, and it is perhaps among the first to be explicitly demonstrated in a publicly-available software simulation.

To construct a comprehensive and detailed simulation in the absence of a fully developed model, two separate strategies were employed. First, high resolution structures of translational components were aligned to locations that are defined in low-resolution models. Second, the simulation is built around a 'core' set of components and events that consist of the mRNA, tRNAs and translation factors, as located at the interface between

the two ribosomal subunits. The detailed movements of the global structure of the two ribosomal subunits are not currently represented. Instead, small regions of the ribosomal subunits that are in immediate proximity to the core components are highlighted during the simulation. The constraints imposed by the core components upon these ribosomal regions then form the basis for the global conformational changes of the subunits that must occur during the translational cycle.

The simulation consists of approximately 80 distinct steps, although a certain fraction of these are concerned with presentational events, as opposed to actual model events. Because there are a large number of intricate components and events in the process, the script makes extensive use of operations that change the viewpoint and graphical representation of molecules in order to optimally present particular details to the observer. The structural content includes atomic models of the small and large subunits, tRNAs, mRNA, and translation factors EF-Tu and EF-G. All the models are loaded into the program and aligned to a common geometrical reference frame. A screen shot of the simulation displaying a subset of the models is shown in Figure 49.

The structural models are taken from snapshots of the ribosome in various functional states. Some of the movements in the simulation, such as docking of translation factors, are represented by simple geometrical interpolation of the static structures from one position and orientation to another. Other events, such as codon-anticodon recognition, peptide bond formation and translocation, are represented by playback of molecular dynamics movies that were produced from SMD simulations.

An associated html document can be loaded into a web browser, which will then provide a collection of runtime controls that allow the user to single-step through the simulation

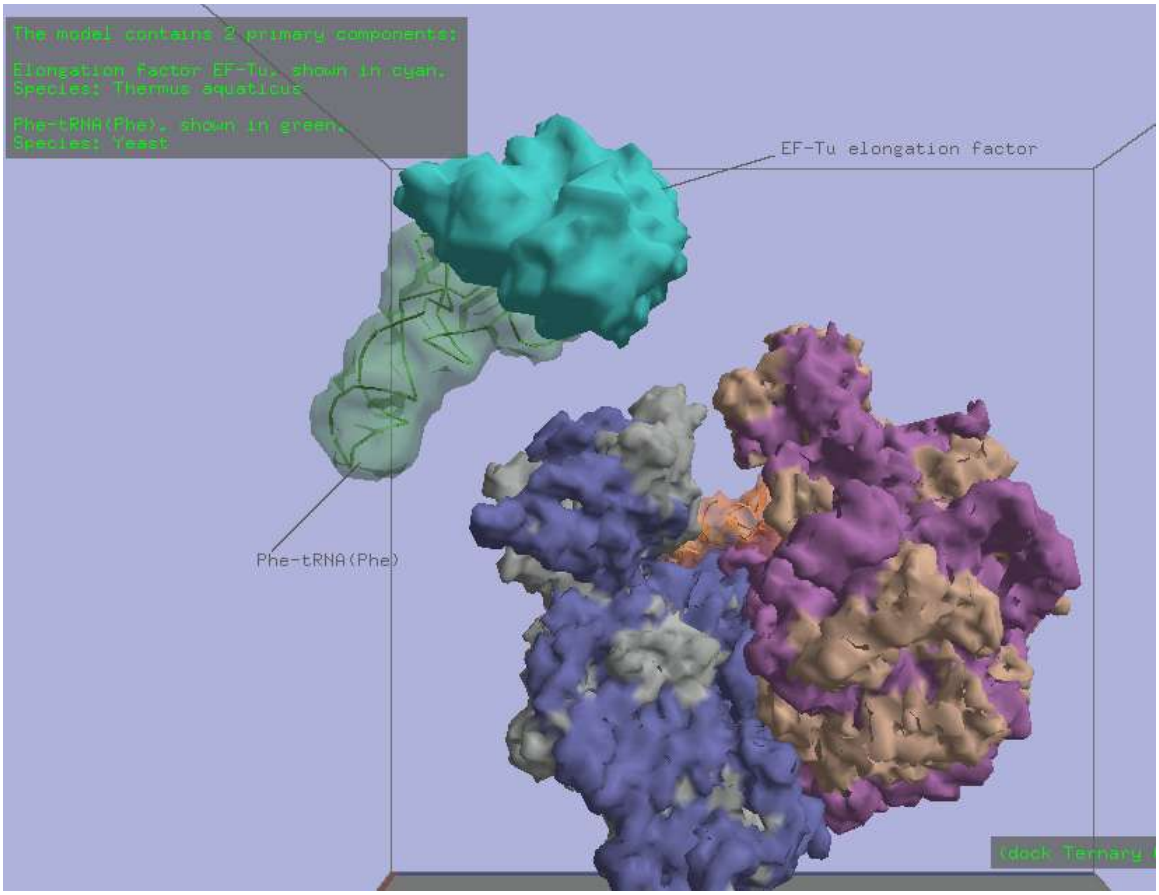

Fig. 49. Screenshot from the schematic simulation of the elongation cycle of the ribosome. The image shown occurs at the end of step 15, which is named 'label TC Components'.

or jump directly to any particular event. The simulation is not a pre-recorded animation. At any point, the user may stop the playback and use the 3D interface controls to change the view point and inspect the structural models. This 'control panel' html document is available through the 'Demos' link on the index html page of the program documentation. The actual control panel html document and the files that define the schematic simulation are both located in the exp/sim/eln001 subdirectory.

18.1.1. Outline of the Simulation

The content and supporting context of this simulation will be discussed in four sections: 1) initial setup, 2) decoding, 3) peptide transfer, and 4) translocation. The initial setup section discusses the construction and placement of the models that define the context for the beginning of the elongation cycle. The remaining sections reflect the current conceptual division of the elongation cycle into three distinct phases. Each of these phases is in turn subdivided into a number of distinct substeps.

18.2. Initial Setup

18.2.1. Placement of Small and Large Subunits

The location of the models for the small and large subunits (1J5E and 1FFK) were done by aligning them to the corresponding subunits in the low-resolution structure of the 70S ribosome, (1GIX and 1GIY) (Yusupov et al., 2001), using the helix alignment algorithm of the program. The small subunit model 1J5E was derived from an x-ray crystal structure of *Thermus thermophilus* at 3 A resolution (Wimberly et al., 2000). The original pdb code for this model was 1FJF. The large subunit model 1FFK is from (Ban et al., 2000).

18.2.2. Development and Initial Placement of P-site tRNA

In addition to the two subunits, the Yusupov model also locates three tRNA models in the A, P, and E sites. These models are only low-resolution placeholders, and do not reflect the detailed conformation of the tRNAs. This schematic simulation was intended to

represent the formation of the first peptide bond, which occurs between the fMet-tRNA (fMet) initiator tRNA in the P-site and an elongator tRNA in the A-site, Phe-tRNA(Phe). Since a high-resolution model of an initiator tRNA docked in the P-site was not available, the tRNA from pdb model 2FMT was used (Schmitt et al., 1998). This is a crystal structure of *E. coli* formyl-methionyl-tRNA(fMet) complexed with its transformylase enzyme at 2.8 A resolution. The acceptor stem of the tRNA in this model deviates from free tRNA as a consequence of binding to the formylase. To adjust for this, an SMD simulation was done to align the model more closely to the bound P-site conformation on the ribosome. The first step was a geometrical alignment of the anti-codon stem loop (ASL) of 2FMT.D to that of 1GIX.C. Then, in the SMD simulation, movement was disabled for the lower portion of the ASL of 2FMT.D. A torsion spring script was defined to align the backbone torsion angles of the 3' terminal residues of 2FMT.D to the corresponding angles of 1GIX.C. In addition, a backbone translational alignment script was run to align the position of the backbone atoms of the 2FMT.D terminal residues to the corresponding residues of 1GIX.C. The simulation was able to successfully align the model to the target so that backbone atom deviation was less than 0.5 A and torsion angle deviation was less than 5 degrees.

Although the anti-codon residues of 1GIX.C are well aligned with the codon in the 30S P-site, the conformation of the 3' end of 1GIX.C in the 50S P-site deviates from the conformation for peptide transfer that is predicted by crystal models of substrate analogs. This is to be expected, as the 1GIX.C structure is actually a deacylated tRNA(Phe), not the initiator molecule fMet-tRNA(fMet). Consequently, the A76 residue of 1GIX.C projects into the 50S P-site binding pocket, in contrast to the position of the analogous

residue in the CCA chain of pdb model 1M90 (Hansen et al., 2002), which was used to define the conformation for peptide transfer. Another SMD simulation was therefore done to align the 3' end of the initiator tRNA to that of the CCA chain of model 1M90. The resulting model was saved as a new static PDB structure (named 'align369_iaa_1') and will be referred to subsequently as the 'initiator tRNA'.

18.2.3. Development and Initial Placement of mRNA

The mRNA used in the simulation is a full atomic resolution 27 residue model that was wrapped around the path defined by pdb model 1JGO (Yusupova et al., 2001) as described in the mRNA wrap application above. An additional forcefield simulation was then done to align a portion of this model with the P-site tRNA, in which base pairs were formed between the P-site codon AUG and the anticodon CAU.

18.2.4. Presentation of Initial Configuration

The first 13 steps of the simulation are concerned with the presentation of the initial components and configuration of the elongation cycle. When the simulation is run for the first time in a session, there is a short delay in which all models in the simulation are loaded and initialized. This one-time setup avoids unnecessary delays during subsequent executions in the simulation session. Then, after an optional timing calibration step, the 30S and 50S subunits are presented and briefly described. A closer view of the mRNA bound to the 30S subunit is shown, and then the initiator tRNA is displayed in its docked location between the 30S and 50S subunits. Then a closeup view shows the base-pairing between the codon and anticodon in the P-site.

18.3. Decoding

The decoding phase of the elongation cycle is concerned with the selection of an aminoacylated tRNA that is a correct match for the current mRNA codon in the A-site of the ribosome. The precise mechanism for how this is done is still not yet known, although a great deal of exciting progress has been made in the last few years. A fully satisfactory model for how the decoding process could work has been elusive and perhaps somewhat mysterious. One reason for this has to do with the very small energetic differences between cognate and near-cognate interactions between mRNA and tRNA. If only the hydrogen bonds between the Watson-Crick base pairs are considered, then energy differences can be just a few kcal/mol. This would imply error rates of amino acid incorporation on the order of 10^{-2}, but the actual error rates in vivo are much smaller, in the range of 10^{-3} to 10^{-4} (Rodnina and Wintermeyer, 2001).

A number of decoding models have been proposed over the years that try to account for the increased accuracy. Two main proposals have been the 'kinetic proofreading' model and the 'distortion detection' model. The kinetic proofreading model, also called the 'two step' model, works to amplify the energetic difference between cognate and near-cognate tRNAs through a sequential process. There is an initial selection step, in which the ternary complex makes contact with the mRNA in the A-site, followed by GTP hydrolysis, and then a second 'proofreading' step in which the tRNA interaction is tested

for a second time, to be followed either by acceptance or rejection (Rodnina and Wintermeyer, 2001).

The 'distortion detection' model proposes that amplification of the energetic differences occurs through the actions of a decoding center in the ribosome. The main idea is that near-cognate or non-cognate codon-anticodon duplexes create distortions of the canonical A-form helix. These distortions then result in broken or changed hydrogen bonding patterns in the region around the duplex, which occur in addition to the direct Watson-Crick base pairing. These changes affect residues of the ribosome surrounding the codon-anticodon duplex, resulting in a structural amplification of the energetic differences (Lim and Curran, 2001), (Ogle et al., 2003).

There is compelling experimental evidence for the kinetic proofreading model. Single molecule FRET experiments have shown distinct temporal states that occur on delivery of tRNA to the ribosome and additional subsequent states that are associated with GTP hydrolysis regulated by EF-Tu (Blanchard et al., 2004). There is also strong evidence for the distortion detection model, including crystal structures that show interactions between the codon-anticodon duplex and the 16S rRNA nucleotides G530, A1492, and A1493 (Ogle et al., 2001). The dichotomy between these two types of decoding models may be a reflection of different perspectives that develop from the two research methods that have been used. The kinetic proofreading model emerged mostly from kinetics experiments and the distortion detection model relies primarily on structural data. As has been noted recently (Ramakrishnan, 2002), the two models are not mutually contradictory and it is likely that both are involved in explaining the accuracy of decoding.

Biochemical and kinetic studies of decoding have continued to characterize it as a succession of intermediate states. One recent review broke down the decoding process into the following 6 steps: 1) initial binding, 2) codon recognition, 3) GTPase activation, 4) GTP hydrolysis, 5) EF-Tu conformational change and 6) accommodation (Rodnina et al., 2005). The elongation simulation of the Ribosome Builder has represented this sequence in varying degrees of detail. Some of the steps, such as codon recognition, are represented in greater detail, with presentation of multiple conformational changes. Other steps are indicated more schematically, such as GTP hydrolysis, or only implicitly referred to, such as GTPase activation. From initial binding of the ternary complex to the completion of the accommodation step, there are currently a total of 27 model and presentational steps in the simulation.

18.3.1. Initial Binding of the Ternary Complex

The presentation of the decoding phase begins with the display of a model of the ternary complex located at a small distance from the ribosomal complex. The model consists of Phe-tRNA(Phe), EF-Tu and GDPNP, with PDB code 1TTT (Nissen et al., 1995). This is a structure from *T. aquaticus* at 2.7 A resolution. The conformation of EF-Tu is an approximation of the GTP-bound state through the use of the non-hydrolyzable analog GDPNP.

There are currently no high-resolution crystal structures of the ternary complex bound to the ribosome, although there have been ongoing improvements in cryoEM structures, with a 13 A model of *E. coli* (Stark et al., 2002) followed by a 9 A model (Valle et al., 2003b). In the absence of detailed coordinate data, the initial binding position of the

ternary complex in the current elongation simulation is just a rough approximation. The location was defined by first aligning the anticodon stem loop (ASL) of the EF-Tu-bound tRNA to the corresponding ASL of tRNA 1GIX.B docked in the A-site. The position of the ternary complex was then manually adjusted to reduce steric conflicts with the 30S and 50S subunits. The results can be observed at the end of step 16, 'dock Ternary Complex'.

The emerging cryoEM data on the location of the ternary complex is expected to substantially alter the existing hypothetical position shown in the current simulation, particularly the positioning of the Sarcin-Ricin Loop (SRL) on the 50S ribosome with respect to EF-Tu. In the model of (Valle et al., 2003b) the SRL is in much closer proximity to the GTP binding pocket of EF-Tu, while in the current simulation it is closer to domain II. In order to effect this improved positioning however, additional structural remodeling of the 50S L7/L12 stalk and related regions will need to be done in order to achieve steric consistency.

18.3.2. Codon Recognition

After binding of the ternary complex, the simulation moves to a closeup view of the codon-anticodon interaction in the A-site in preparation for the process of codon recognition. This is followed by a hypothetical SMD simulation in which the anticodon in the ASL of the incoming tRNA undergoes a conformational change to base pair with the mRNA codon, as shown in Figure 50. The flexibility of the ASL is evident in the simulation as base pairing can occur without corresponding changes in position of either the mRNA codon or of the EF-Tu bound portion of the tRNA. It is possible that global

conformational changes within various parts of the 30S and 50S subunits might enable simultaneous movements of these other regions as well during codon recognition. However, the demonstrated flexibility of the ASL indicates that initial base-pairing could occur with just the minimal movements of the ASL itself.

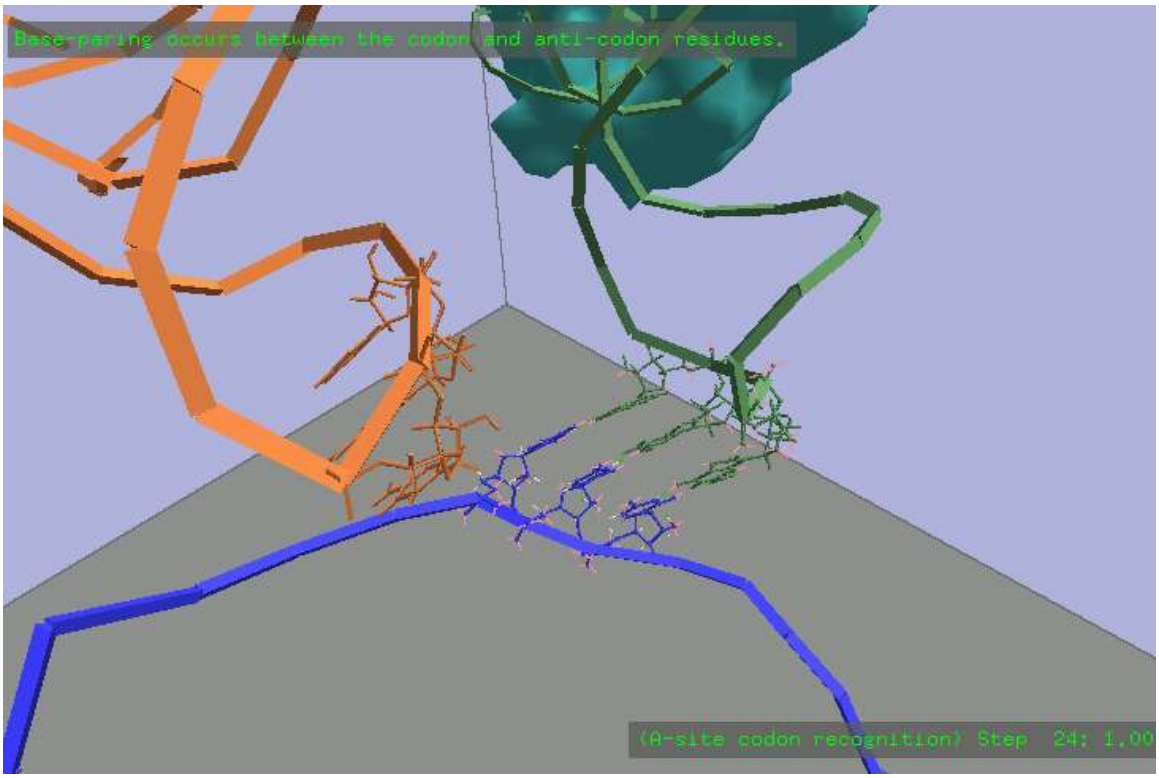

Fig. 50. Screenshot of A-site codon recognition. Three base pairs are formed between the mRNA codon UUU and the Phe-tRNA(Phe) anticodon GAA.

The view then shifts to a presentation of a portion of the ribosomal decoding center that responds to the formation of the codon-anticodon duplex. As discussed in (Ogle et al., 2003), a mechanism has been proposed which involves the sensing of the minor groove of the codon-anticodon helix by ribosomal 16S residues A1492, A1493 and G530. Support for this hypothesis exists in the crystal model 1FJG (Carter et al., 2000), which is a structure of the 30S subunit combined with the antibiotic paromomycin. In this

structure, the bases A1492 and A1493 are 'flipped out' from their normal helical conformations within helix 44 and they are in positions that would allow them to act as sensors, as proposed above in the 'distortion detection' model.

An SMD simulation to represent the flipping motion of these two residues was performed on a subset of helix 44 in the 30S model 1J5E. A segment of 8 dynamic residues was created, G1489-C1496. The target conformation was determined by the residues A1492 and A1493 in the 1FJG model, which was loaded and aligned with 1J5E, using helices 13 and 2. To allow for small conformational changes at the end of the dynamic segment, a spring tether script was defined to maintain the backbone atoms of the endpoint residues close to their initial locations. A backbone torsion spring script was defined to drive the central 4 residues to the backbone conformation of the corresponding residues in the target. In addition, backbone atom translational forces were defined to induce the alignment of the backbone atom positions of the 4 central residues to the target positions. Finally, a glycosidic torsion angle torque was needed to produce the correct orientation of the base of A1492. This may have been because the codon-anticodon residues were not included in this SMD simulation.

Currently, this SMD simulation acts primarily as a confirmation of the proposed flexibility and possible movement of the A1492 and A1493 residues proposed by (Ogle et al., 2003). The simulation should be extended at the very least to include the interactions with the codon-anticodon duplex. Inclusion of additional surrounding regions of the decoding center should also be done. It is anticipated that the highly defined and artificial conformational changes imposed on the small 8 residue dynamic segment may result in additional concerted movements in the adjacent regions of the decoding center.

18.3.3. GTP Hydrolysis

Upon the completion of the codon recognition event, the simulation view shifts out again to show the decoding center in relation to EF-Tu and the bound GTP analog. The signal from codon recognition in the decoding center on the 30S subunit is somehow presumed to communicate a signal over to the 50S side of the ribosome, where a process of GTPase activation takes place to induce the hydrolysis of the bound GTP on EF-Tu. The mechanism by which such a signal is communicated is not yet known. Various conformational changes that result upon codon recognition are likely candidates, including the domain closures that have been observed in the 30S subunit (Ogle et al., 2002) and bending observed in the A-site tRNA (Frank et al., 2005).

Because it cycles between a GTP-bound conformation and a GDP-bound conformation, EF-Tu is a G-protein. G-proteins are an important class of molecule used in many regulatory and signaling processes in the cell. The translation factors IF2, EF-Tu, EF-G, and RF3 are all G proteins that act to catalyze a number of the steps in protein translation (Marzi et al., 2003), (Zavialov and Ehrenberg, 2003). The G proteins involved in signaling include the family of small Ras-related proteins and the heterotrimeric G proteins with α, β and γ subunits. All G-proteins share a common structure call the G-domain that functions as a switch. This structure consists of a number of conserved features including a nucleotide binding pocket, the P-Loop, and two additional loops called switch I and switch II. The switch loops form hydrogen bonds to the gamma phosphate of the bound GTP residue and function as a loaded spring mechanism. Release

of the phosphate upon hydrolysis allows the switch regions to relax into the GDP conformation (Vetter and Wittinghofer, 2001).

The exact mechanism of GTPase activation by the ribosome is not yet known (Rodnina et al., 2005). Two regions on the 50S subunit have been implicated in the stimulation of GTP hydrolysis on the translation factors. The first of the ribosomal regions is the factor binding site consisting of proteins L7/L12, L10, L11 and the L11-binding region of 23S rRNA (Mohr et al., 2002). The second region is the Sarcin-Ricin Loop (SRL) in helix 95 of 23S rRNA (Blanchard et al., 2004). A mechanism for GTPase activation has been recently proposed which involves a conformational change of the GTPase-associated Center (GAC) of L11 and its associated rRNA (Valle et al., 2003b). However, this region was not resolved in the 1FFK model used in the schematic simulation. Instead, the changes leading to GTPase activation and hydrolysis of the GTP residue in the simulation are indicated in symbolic form by the brief flashing of the GTP residue. This is followed by the dissociation of the free phosphate molecule, leaving a GDP residue in the binding pocket. Then the change in conformation of EF-Tu to the GDP bound state is indicated by a second symbolic flashing, in this case, of the entire EF-Tu structure. This is followed by dissociation of EF-Tu from the ribosome.

18.3.4. Accommodation

Accommodation is the final step in the decoding phase, in which the 3' end of the A-site tRNA dissociates from EF-Tu and moves into the Peptidyl Transferase Center (PTC) in the 50S subunit. Although the dissociation of EF-Tu from the ribosome is shown as a preceding event in the simulation, in practice there must be some degree of simultaneous

movement of both the 3' arm of the tRNA and EF-Tu as they dissociate from each other. In accordance with the proofreading action proposed in the kinetic proofreading model, the process of accommodation may be also be replaced by the rejection step in which a non-cognate tRNA is dissociated from the ribosome. The current representation of accommodation in the simulation is defined by a simple linear interpolation of the tRNA from the docked location of 1TTT.D on EF-Tu to the docked location of 1GIX.B on the 70S ribosome. Once the 3' arm of the aminoacyl-tRNA is fully accommodated within the PTC, it is positioned in conjunction with the P-site tRNA for participation in the peptide transfer reaction.

18.4. Peptide Transfer

Peptide transfer is conceptually a more concise process than the other two phases in the elongation cycle, which have been divided into numerous distinct substeps. In spite of the disparity however, the event of peptide transfer is *the* singular event of the elongation cycle. It can be considered as the central event that is supported by the others preceding and following it. The addition of an amino acid onto the growing peptide is the ultimate purpose of the elongation activity. For this reason, and also because the precise boundaries separating the three phases still remain to be fully clarified, the peptide transfer reaction is used to designate the central third phase of the elongation cycle.

The reaction of peptide bond transfer in the ribosome involves a nucleophilic attack by the α nitrogen of the A-site aminoacyl residue upon the carbonyl carbon at the ester bond between the P-site tRNA and the nascent peptide. Compared to the uncatalyzed reaction, the ribosome accelerates peptide bond formation by more than 10^5 (Rodnina and

Wintermeyer, 2003). The atomic resolution structure of the 50S subunit has shown that the closest protein to the Peptidyl Transferase Center (PTC) is at least 18 A away (Nissen et al., 2000), which means that the catalytic action must be determined instead by the surrounding ribosomal RNA. This has established the ribosome as a ribozyme. However, although the immediate participants in the reaction have been identified, the precise mechanism by which the reaction is catalyzed is not yet fully known.

Biochemical studies of peptide bond formation go back a long way, and include the pioneering work of Monroe et al. in the 1960s. They established that the 50S subunit alone was responsible for catalyzing the reaction. In their studies, the antibiotic puromycin was used to terminate protein synthesis and release the nascent peptide from the ribosome. It did this by functioning as an A-site substrate mimic. After forming a peptide bond to the nascent peptide in the P-site, further elongation was prevented by the amide bond between the adenine and aminoacyl groups in puromycin. The amide bond causes the carbonyl carbon to be less susceptible to subsequent attack in comparison to the normal ester bond between the 3' oxygen of tRNA and the carbonyl carbon of the aminoacyl residue (Maden, 2003).

A number of excellent reviews summarize the recent work done towards deciphering the mechanism of peptide transfer, including (Moore and Steitz, 2003b), (Rodnina and Wintermeyer, 2003), and (Steitz, 2005). The basis for much of this work began with the structural models of the 50S subunit co-crystallized with A and P site substrate analogs in (Nissen et al., 2000). In one of their models, PDB code 1FFZ, the Yarus inhibitor was used. This inhibitor is a transition-state analog of the peptide transfer reaction that consists of a CCdA fragment linked via a phosphoramide bond to puromycin (Welch et

al., 1995). In the structure, the CCdA portion binds in the P-site as an analog of the peptidyl tRNA and the puromycin portion binds in A-site. An associated model in the paper, PDB code 1FG0, defines an additional portion of the A-site tRNA 3' arm that overlaps with the puromycin fragment of 1FFZ. A subsequent structural model, PDB code 1M90, provided an additional piece of the puzzle by resolving a tRNA analog with peptidyl fragment in the P-site by itself, through the use of sparsomycin (Hansen et al., 2002).

This structural data helped to motivate a number of proposals for hypothetical elements in the catalytic mechanism. These elements can be divided into two basic categories. The first category consists of elements that assist in the structural formation of the transition state of the reaction. The second category consists of elements that act as general acid/base catalysts for exchanging protons with the substrates during the reaction. From analysis of the structures above, it was observed that the N3 and 2'O atoms of residue A2451 of the 23S rRNA and the 2'O atom of the terminal adenine residue of the P-site tRNA were close enough to contribute to a hydrogen bonding network with the substrates. In addition, (Nissen et al., 2000) proposed that the N3 atom of A2451 might act as a general base to extract a hydrogen from the alpha amino group to catalyze the bond reaction.

In order for N3 of A2451 to act as a general base catalyst, its pKa value would have to be shifted significantly upwards from the normal unperturbed value of around 4. Chemical probing evidence for this shift was presented by (Muth et al., 2000), and subsequent work by (Katunin et al., 2002) provided additional mutational and kinetic evidence for a catalytic contribution by A2451. However, a number of other studies cast doubt on the

question. Mutational and chemical probing analysis indicated that the observed shifted pK_a may have resulted from conformational changes in the structure (Muth et al., 2001). Alterations in the chemical probing conditions revealed that the perturbed pKa conformations were different from those expected to be found in active ribosomes (Bayfield et al., 2001). Other experiments provided evidence that substrate analogs are more strongly affected than full-length tRNAs by some of the proposed catalytic elements (Youngman et al., 2004), thus possibly invalidating some of the conclusions that can be drawn from results obtained with these substrate analogs.

As can be gathered from the intensive research described above, discovery of the mechanism of peptide transfer may prove to be as great a challenge as that of the decoding process. The current Ribosome Builder simulation of peptide transfer only investigates the most basic catalytic contribution by the ribosome, which is a positional one. In the simulation, the substrates are initially placed at positions and orientations that optimize the occurrence of the subsequent reaction. It is expected that this basic simulation will serve as a three-dimensional dynamic framework upon which additional elements will subsequently be added. Such additional elements might refine the existing positional catalysis, perhaps through the addition of concerted movements by parts of the ribosome. Other elements, such as residues which donate and accept protons, could be added for investigating more complex catalytic mechanisms.

18.4.1. Development and Placement of A-site tRNA

In the current accommodation step of the simulation, the model of the A-site tRNA that is associated with EF-Tu is simply replaced by a model of the tRNA that has already been

accommodated within the A-site of the 50S subunit. The movement of the model into the 50S is done through a simple rigid-body linear interpolation from its initial position of alignment with EF-Tu into the final accommodated location. As with the initial placement of the P-site tRNA model 1GIX.C described above, the A-site tRNA model 1GIX.B had to be remodeled into a conformation that was stereochemically appropriate for peptide bond formation. First, a phenylalanine residue was grafted on to the 3' end of the tRNA using PDB model 2FMT as a template. Then, the PDB model 1FGO from (Hansen et al., 2002) was aligned to the 50S model 1FFK using the 'AlignSelectedResidues.lua' script with residue A2517 selected. A 6 residue dynamic segment was created at the 3' end of 1GIX.B. This segment was aligned to the 1FGO fragment with an SMD script that applied spring forces between the corresponding atoms C2*, C3* and C4* of the residues 1GIX.B.A76 and 1FGO.PPU76.

18.4.2. The Peptidyl Transferase Center (PTC)

The simulation of the peptide transfer phase of elongation currently consists of 16 steps. The majority of these are presentational steps that display various parts of the Peptidyl Transferase Center (PTC) that are relevant to the actual peptide bond formation event. The simulation begins by zooming in to show the 3' arms of the A and P-site tRNAs. This is followed by a display of the surface representation of the surrounding 23S rRNA residues that constitute the inner shell of the PTC and form a binding pocket for the tRNA 3' arms. The residues in the PTC pocket were defined using the script 'OutputChainResAtDist.lua' with the 3 terminal residues of both the A and P site tRNAs selected (1GIX.B, 1GIX.C), with a distance threshold of 20 A and the target chain as

1FFK.0. Residues A2602 (1FFK num A2637) and U2585 (1FFK num U2620) were excluded from the pocket model so that they could be displayed independently from the pocket surface. The surface was created at 1.3 A resolution.

After the pocket surface is displayed, it is made partially transparent to show two important features in its internal structure. These are the A and P loops of the PTC, which interact with and help to orient the tRNA arms through the formation of several base pairs (Blanchard and Puglisi, 2001), (Nissen et al., 2000). Then, the conserved residue A2602 is highlighted in the context of the opening of the tunnel at the bottom of the PTC pocket. This residue undergoes a change in position during translocation and may function as a conformational switch (Bashan et al., 2003).

18.4.3. The Peptide Bond Reaction

The state just prior to the actual peptide bond reaction is shown in Figure 51. The attacking nitrogen atom is shown in the free amine state, having already lost the third hydrogen atom that would normally be bonded to it in the ammonium state at neutral pH. The simulation of the reaction begins with the movement of the nitrogen towards the carbonyl carbon at the ester bond. An external spring force constraint was applied between the two atoms to induce the approach. As they approach, small resulting movements in the backbones of both tRNA acceptor arms is evident. When the two atoms are close enough to form a covalent bond, the bond structure is changed to reflect this, and one of the two remaining hydrogens is removed from the nitrogen. In actuality, a temporary tetrahedral intermediate state of the nitrogen is proposed to exist, which would allow both hydrogens to remain bonded to it for a short time (Green and Lorsch, 2002).

One of the hydrogens would then be removed, by a mechanism that has yet to be been determined.

Momentarily after the new bond is formed in the simulation, the bond between the 3' oxygen of the P-site tRNA and the carbonyl carbon is broken, leaving a deacylated tRNA in the P-site. At this point, proton rearrangement would lead to the addition of a hydrogen on the 3' oxygen, but this is not currently shown in the simulation. During the rest of the simulation step, a new external constraint is applied to the newly-formed peptide bond to induce a rotation to the normal planar conformation.

The movements and events in this simulation step can be difficult to observe when the simulation is played at the normal rate. When it is desired to inspect the activity more slowly and in greater detail, this is made possible by using the 'single tick' option in the Annotation Step Control panel of the associated HTML document. When this option is turned on, the simulation must be manually incremented for each interpolation tick of the step by clicking the 'Run' control.

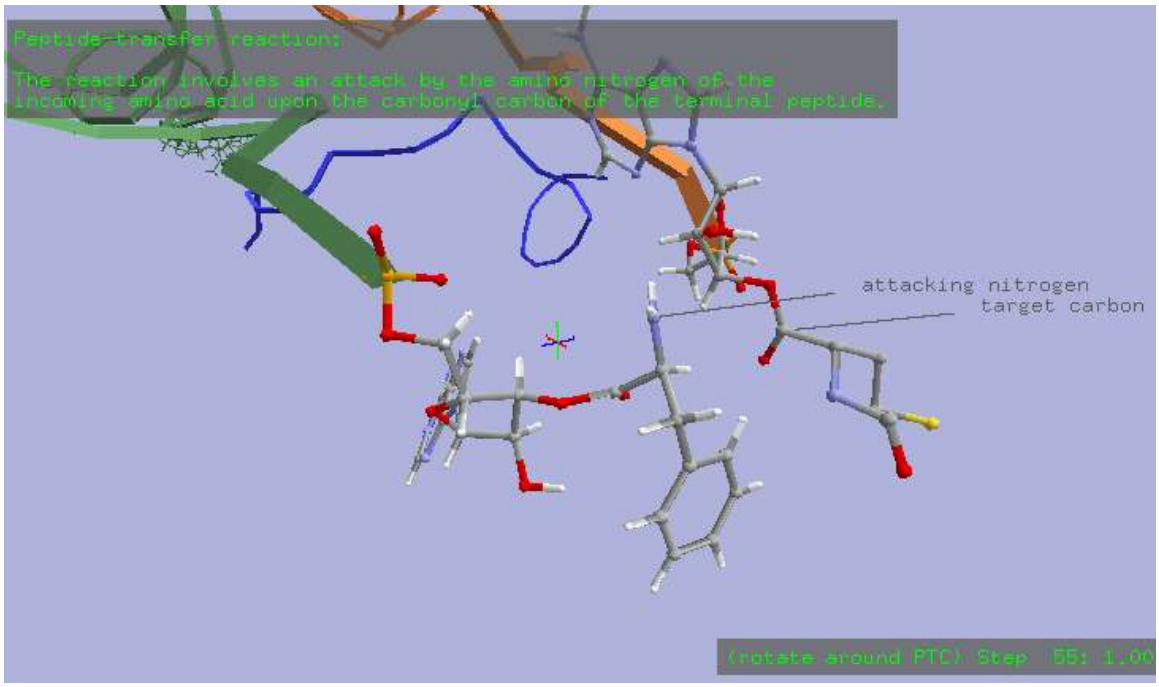

Fig. 51. Screenshot of peptide transfer reaction in the Peptidyl Transferase Center (PTC) of the 50S subunit. A phenylalanine residue is in the A-site and a formyl-methionine residue is in the P-site.

18.5. Translocation

Translocation is the final phase of elongation that is represented in the simulation. After the transfer of the peptide to the A-site tRNA, translocation is the process by which the two tRNAs and the mRNA must move ahead in order to produce a vacant A-site and allow for another cycle of elongation. In addition to the A and P sites, an E site has been identified in the ribosome and the P-site tRNA will move into this E site as the A-site tRNA moves into the P-site. The translocational movements and the sequence in which they happen are still not known with certainty, and a number of alternative models have been proposed. A number of aspects must be accounted for in the process including the question of whether the tRNAs are translocated simultaneously at both ends or not.

Another question is how portions of the ribosome move with respect to the tRNAs and mRNA during the process. The process is also catalyzed by the translation factor EF-G and there is still much to be learned about where it binds to the ribosome and exactly when and how it is activated, hydrolyzed and released.

One of the most notable aspects of translocation in comparison to the other phases of elongation is the very large conformational change that must occur. Collectively, the tRNAs and the mRNA must move by approximately 20 A through the ribosome (Southworth and Green, 2003). CryoEM maps of the ribosome in states preceding and following translocation revealed a 'ratchet-like' motion of the subunits in which the 30S subunit undergoes a counterclockwise rotation of about 6 ° relative to the 50S subunit and then partially rotates back by 3 °. The first rotation was associated with EF-G binding and the second rotation was associated with GTP hydrolysis (Frank and Agrawal, 2000). Subsequent cryoEM structures indicated that an enabling condition for the ratchet motion was a deacylated tRNA in the P-site, indicating that the presence of the peptide acts as a 'lock' on the motion (Valle et al., 2003a).

18.5.1. Docking of EF-G on the Ribosome

The starting model for EF-G in the Ribosome Builder simulation was taken from a crystal structure of EF-G in the GDP conformation at 2.4 A resolution from *T. thermolphilus* (Al Karadaghi et al., 1996), which was the highest resolution model available at the time. The simulation begins by displaying this model at a small distance away from the ribosome. The docking of the model is done with a rigid body linear interpolation to a location on

the ribosome determined by (Valle et al., 2003a). In this work, a crystal structure of a mutant EF-G complexed with GDP (PDB code 1FNM) was taken from (Laurberg et al., 2000) and docked into the cryoEM map of the 70S ribosome complexed with EF-G-GDPNP in the GTP conformation. Domains I and II of the EF-G crystal structure appear to align well with the cryoEM density, as seen in Fig. 4 of (Valle et al., 2003a). Rigid-body docking of domains III, IV and V was done by (Valle et al., 2003a) to produce a new low-resolution model, 1PN6. This 1PN6 model was aligned to the standard reference frame in the Ribosome Builder simulation to provide the target docking location of the 1DAR model, which was determined by aligning 1DAR to 1PN6 using residue ILE21. In the docked conformation, the GTP residue in the nucleotide binding pocket of EF-G is closely positioned near the Sarcin-Ricin Loop (SRL) of the 50S subunit, indicating that the docking location is plausible. Even though a model in the GDP conformation was used, the portion that overlaps with the EF-G-GTP conformation consists of domains I and II, which are in contact with the 50S subunit. The portion of EF-G that appears to change consists of domains III, IV and V, which appear to undergo an arc-like rotation between the EF-G-GTP state and the EF-G-GDP state upon hydrolysis. As these domains are in contact with the 30S subunit, this may be structural evidence for the mechanism of the 3 ° return rotation of the 30S that is associated with GTP hydrolysis (Valle et al., 2003a).

18.5.2. GTP Hydrolysis

After EF-G is docked on the 70S complex, the view shifts to a closeup of the mRNA, tRNAs and EF-G with the 50S subunit in the background. The 50S subunit surface is

faded away to reveal the SRL (helix 95) of the 23S rRNA in close proximity to the GTP-bound residue on EF-G. A flashing of the SRL is done to indicate the GTP-ase activation of GTP hydrolysis, followed by cleavage and release of the phosphate residue from the binding pocket.

18.5.3. Conformational Change of mRNA

As a result of the induced conformational changes (not currently shown) resulting from EF-G binding and the subsequent GTP hydrolysis, the translocation of the mRNA and the tRNAs is presented as the next event in the simulation, as seen in Figure 52. The A and P-site tRNAs translocations are done using rigid-body linear interpolations to the subsequent P and E sites. The mRNA translocation is represented by a recorded SMD simulation in which external positional spring forces and torsion-angle torques were applied to a 21 residue subset of the 27 residue mRNA chain. In the SMD simulation, each of the 21 residues is driven towards a target conformation defined by a residue located 3 positions upstream from it, in accordance with a 3-residue translocation. In conjunction with the mRNA and the tRNAs, the EF-G undergoes a linear interpolation intended to represent a partial movement into the A-site of the 30S subunit.

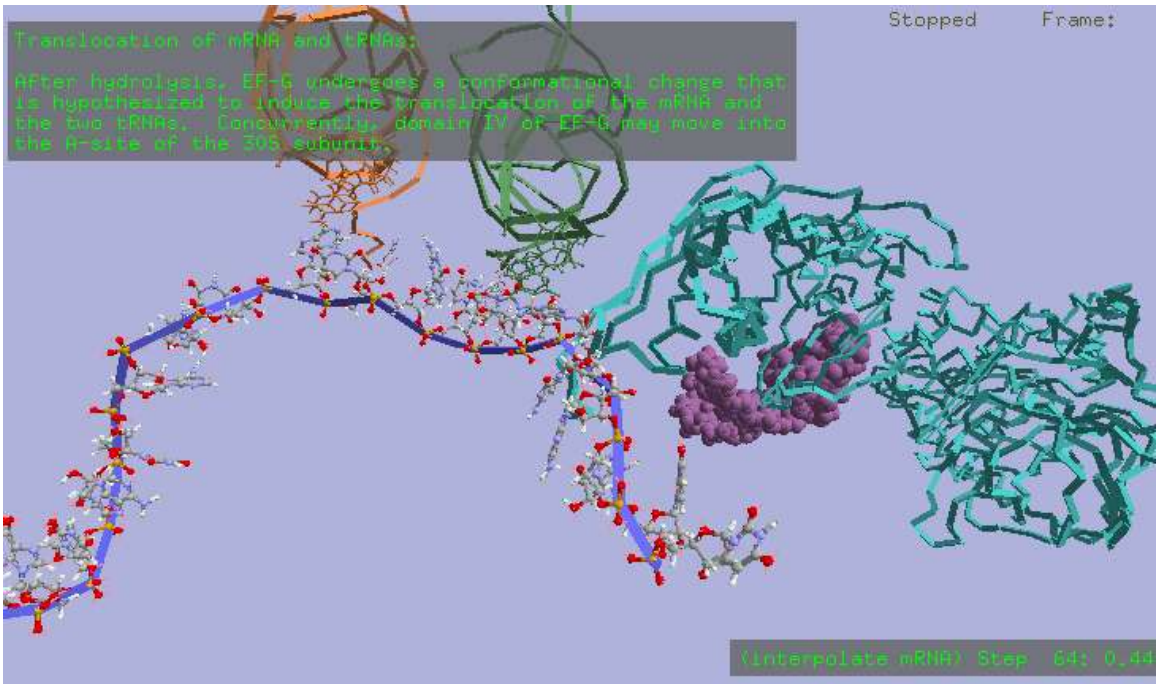

Fig. 52. Screenshot of the translocation step. The mRNA is shown in ball-stick representation with a blue backbone. The P-site tRNA is shown in orange, the A-site tRNA is shown green, EF-G is shown in cyan and the Sarcin-Ricin Loop (SRL) of the 23S rRNA is shown in purple.

18.6. Conclusion

After the completion of the mRNA and tRNA translocations, the view pulls back to show the subsequent dissociation of EF-G from the ribosome. This is followed by a short display of journal article references that contributed to the content of the simulation. The schematic simulation is obviously quite preliminary and much greater detail must be added to it. However, in order to construct this preliminary simulation, the following structural modeling was done: 1) a structural model of initiator tRNA was created in the P-site, 2) a structural model of tRNA in the A-site was created, in which the 3' arm was positioned to a conformation that enabled it to participate in the PTC reaction, and 3) an atomic resolution model of a 27 nt mRNA was produced in a docked location on the 30S

with A and P-site codons aligned to base pair with the corresponding A-site and P-site tRNA anticodons. In addition to the required structural modeling, three conformational changes of structures were produced from SMD simulations: 1) A1492 and A1493 recognition of codon-anticodon duplex, 2) the peptide transfer reaction, and 3) the translocation of mRNA by three codons. In addition, two hypothetical docking locations were defined: 1) docking location of the ternary complex EF-Tu-GTP-tRNA, and 2) docking location of EF-G.

The definitions of the structural models, conformational changes and docking locations listed above were then linked together as needed by a number of rigid-body linear interpolations to move the models between defined locations. This produced a continuous animated model of the entire elongation process. The model events were then supplemented by a number of presentational events including graphical annotations, changes in molecular graphical representations and viewing motions in order to produce a schematic simulation that could be effectively communicated to the viewer. Finally, the software implementation enables flexible execution and inspection of the simulation with the provision of single-step, single-tick and individual step selection controls.

Conclusion

Summary of Accomplishments

A simulation of the entire process of the elongation cycle of the ribosome has been presented. The simulation has included all of the principal structural components of the ribosome at an atomic level of detail, and covers a long molecular timescale in a way that has not been done previously. In order to produce a simulation of such broad temporal and spatial scope, it has been necessary to create a new software framework to support the difficult demands and new requirements that emerged. In doing so, the work has sacrificed a certain initial degree of depth and detail that is possible with more isolated and specific investigations. However, in developing the tools, methods and simulations to handle the entire process, what has resulted is a more general approach that should prove useful for investigating many other kinds of macromolecular systems.

The specific achievements of the research include the de novo construction of a simple all-atom molecular mechanics forcefield. The forcefield represents the fundamental molecular properties of bond lengths, bond angles and steric exclusions. These basic properties are supplemented by forces to reproduce hydrogen bonding, electrostatic and base stacking interactions. A means for maintaining robust covalent bond geometries in response to strong external influences has been produced through a new algorithm called the 'shadow point method', which is based upon representation of the geometrical orientation of the valence electronic structure of the atoms.

A molecular dynamics engine has been constructed in conjunction with the forcefield. The engine employs a simple integration method that uses the accumulated forces and torques on atoms to update their positions each timestep. The engine combines the internal molecular mechanics forcefield with an option to apply externally-defined forces and torques. This supports the approach of Steered Molecular Dynamics (SMD) for producing simulations of long time-scale molecular activity. The engine also provides the ability to record, save and playback 'movies' of molecular dynamics simulations. The forcefield and dynamics engine have been optimized for performance so that they can efficiently simulate very large models such as the small and large ribosomal subunits, consisting of 50,000 atoms or more. The resulting efficiency also makes it practical to conduct such simulations on existing single-processor desktop computers.

This local computational capability is especially important for the individual researcher engaged in structural molecular modeling and simulation, because it enables rapid prototyping and experimentation. This capability is further enhanced by a powerful and flexible graphical user interface. The interface provides a rich set of menus, toolbars and other types of interface controls. The primary interface is a 3D graphics window that supports navigation and manipulation of objects in the 3D space. A variety of graphical representations are used to display the structural details of multiple molecular models within the window. Selection and manipulation operations enable the creation of multi-component macromolecular structural models. The structures can be saved to disk as new PDB files and subsequently be reloaded and used in the construction of other models. An additional interface is provided via a local HTML browser, which enables convenient control of the application through the use of standard HTML documents.

The program also implements an internal script interpreter that uses the Lua scripting language. Lua is a small, robust, efficient and powerful language with a simple and elegant syntax for appeal to the non-professional programmer. The scripting language is supplemented by a large set of documented API functions and library routines to support automization and customization of the application, modeling operations, and SMD applications. This powerful scripting capability has also enabled the creation of schematic simulations, which is a new methodology that has been proposed for modeling and simulation of complex macromolecular systems. A schematic simulation employs static and dynamic structural models, text and graphical annotations, coordinated movement of the three-dimensional viewpoint, scripted translation and rotation of static models and playback of recorded molecular dynamics trajectories.

The implementation of the software program has been done through the use of object-oriented software design methods. The code has been organized into libraries for modularity and reuse. The core code is written in C++ and is largely cross-platform, with a single current implementation on Windows/PC operating systems. A framework for project management and collaboration has been established through the creation of source code and program documentation, unit tests and version control. The program has been released as an open source project on SourceForge.

A number of applications of the program have been presented. The first application presents a method for aligning ribosomal subunits and defining an absolute frame of reference for the assembly of ribosomal complexes. The next application demonstrates the use of the SMD tools to produce the folding of an 12 residue RNA oligomer to a target tetraloop conformation. This was done through the analysis of RNA rotameric

configurations, and the extension of this approach for the discovery of hypothetical dynamic RNA 'convertamers' is considered. The third application presents another SMD simulation of a 27 residue RNA oligmer that begins in the A-form helical conformation and is then 'wrapped' around a low-resolution path of an mRNA that is docked to the 30S subunit. This results in a full atomic resolution model of an mRNA in this conformation which can serve as a component for investigating further interactions with the ribosome. The final application is a schematic simulation of the complete process of elongation on the ribosome. The simulation currently has a total of 80 distinct steps, which are composed of both modeling and presentational events. All of the principal structural components of the process are included at an atomic level of detail, and a number of hypothetical conformational changes are presented including an element of codon recognition, peptide bond formation and translocation of the mRNA. The model is only a preliminary one, but should serve as an explicit framework upon which additional details can be added and changes made.

In the near future, plans for additional work on the project include elaboration of the existing elongation simulation and the creation of schematic simulations for the initiation and termination stages of translation. An urgent short term goal is to port the program to the GNU/Linux platform, as this is expected to greatly increase the prospects for collaboration and contributions to the project. Longer term goals include the creation of models and simulations of eukaryotic ribosomes and other macromolecular systems such as the transcription complex, DNA replication and spliceosomes. Software objectives include a parallel processing implementation, as the arrival of ubiquitous desktop multiprocessor systems is expected in the next few years. Other software goals include

the integration and use of other open source implementations of forcefields, molecular modeling algorithms and scripting tools. Enhancements to the graphical interface to support increasingly rich multi-user interactions through 3D stereoscopic head-mounted displays, haptic controllers, sound i/o and networking are also anticipated.

Multiscale Modeling

This research has been done at a time when the explosions of genomic, structural, biochemical and other kinds of scientific data are threatening to overwhelm our ability to create theoretical models to explain the data. Advances in NMR, X-ray crystallography and cryo-electron microscopy are beginning to produce a continuum of structures from the atomic to the cellular scale. As a consequence, the separate disciplines of structural molecular biology and structural cell biology must combine to address this unified hierarchy of data. To provide the computational modeling for this new discipline, specific mathematical formalisms and computational frameworks will be needed at different levels in the hierarchy (Harrison, 2004).

Work on such multiscale computational frameworks is already underway. The Virtual Cell project provides a formal framework for modeling biochemical, electrophysiological and transport phenomena in a way that takes into account the three-dimensional geometry of a cell and its subcellular organelles (Slepchenko et al., 2003). The IUPS Physiome project also seeks to develop new computational infrastructures but is even more ambitious in scope as it seeks to model biological systems from the molecular to the organismal level (Hunter, 2004).

One of the most difficult challenges in multiscale modeling will be to simulate the structural and dynamic interactions that occur at the interface between the molecular and macroscopic scales. The work by (Alberts and Odell, 2004) offers an impressive combination of biochemical kinetics and mechanical dynamics for representing realistic bacterial motion and actin tail morphology of the intracellular pathogenic bacterium *Listeria monocytogenes*. In their approach, the actin monomers are modeled implicitly with experimentally-determined biochemical parameters such as concentration, diffusion and association/dissociation rate constants. The actin filaments that are formed from the monomers are modeled explicitly in a mechanical framework. The three-dimensional structure of each filament is represented as a rigid rod, with the addition of parameters to approximate some degree of flexibility. The interactions between the rods and cell surfaces are calculated with Newtonian forces and the resulting dynamics qualitatively reproduce realistic motions and actin tail morphology.

The mechanical framework and informational techniques that have been developed in the Ribosome Builder program have been successfully applied to create models and simulations of macromolecular complexes. The same mechanical framework and techniques can be used for modeling mesoscale phenomena such as organelles and intracellular structural networks. The schematic simulation approach that has been proposed is expected to be especially useful in this area, because of its ability to flexibly prototype and represent a complex scene of structures and dynamic events.

The Special Significance of the Ribosome

This research has been motivated by a desire to gain an understanding of the workings of life at the molecular level, and a greater hope to extend that understanding to macroscopic levels. The details of life at the molecular level are continuing to be revealed in an expanding landscape of exquisitely intricate components and interactions. Within this rich landscape, the ribosome has beckoned with a special significance because it is a central component in the larger system that expresses the genetic information of the cell into its material form. Because of this, the ribosome is essential to the existence, maintenance and replication of the cell, and hence to life itself. It is possible to envision mechanisms for translating genotype to phenotype in ways that differ from the transcriptional and translational systems of existing living cells. For life that predated the emergence of the translational machinery, some kind of different system must have existed by necessity. The RNA World Hypothesis (Joyce, 2002) and other 'Origins of Life' (OOL) theories are concerned with the questions of how the first self-replicating systems came into existence and how they might have worked. One OOL theory for the earliest stages of life hypothesizes the spontaneous emergence of metabolic networks from the properties of autocatalytic sets. This theory attempts to explain how reproduction could develop even prior to the existence of informational linear polymers such as RNA (Kauffman, 1993).

The RNA world hypothesis is concerned with the later question of how modern life may have evolved from an existing RNA-based metabolism. The ability of RNA to serve in the dual roles of information carrier and functional molecule give it the potential to be the progenitor of the current system of DNA-based genomes and protein-based metabolisms. There are many indications of possible remnants from this earlier system in present-day structures and processes. These include the RNA primers used in DNA replication, the self-splicing activity of pre-mRNA and the RNA structural and catalytic components of the ribosome. Consequently, understanding the structure and function of existing RNA-based mechanisms has helped to create theoretical models for explaining the evolution of life from an RNA world to our current one (Poole et al., 1998).

Because of its close connection to the evolution of existing life, RNA is also likely to play an important part in the creation of new, artificial molecules and systems of molecular replication. Evolutionary methods such as SELEX have already been used to construct artificial RNA molecules with many structural and functional properties. This method involves the generation of large populations of RNA with variant sequences followed by multiple rounds of selection and amplification to acquire target sequences that possess some desired property such as ligand binding or catalysis (Tuerk and Gold, 1990), (Wilson and Szostak, 1999).

The ability to design RNA molecules for specific binding or catalysis is quite significant. However, as was discovered long ago, the use of proteins for such tasks enables more powerful and varied capabilities. Therefore the next step beyond binding and catalysis is the artificial design of RNA molecules to perform the functions of programmable molecular assembly. The possibility of such systems has already been demonstrated by

the work of (Liao and Seeman, 2004). They have constructed a molecular machine composed of DNA, which was used because it is currently more tractable for design than RNA. This machine consists of small cylinders of DNA that can be programmed, through the addition of external sequence-specific oligomers, to adopt one of two precisely-defined conformations. Then, the particular conformation that has been specified enables the sequence-specific ligation of a product molecule, which in the initial system is also a DNA oligomer. It is important to note that the significance of the machine is not just in its ability to programmably ligate sequences of DNA. There are many macroscale DNA synthesizers in existence which are far more capable. The significance is that the machine is both programmable and is itself nanoscale in size. Like the ribosome, it may be defined as a 'Nanoscale Programmable Molecular Assembler' (NPMA).

The three combined properties of nanoscale dimension, programmability and molecular assembly point the way towards the construction of artificially designed self-replicating systems of molecules. It is the nanoscale nature of the DNA ligation machine that enables the possibility for the machine to be produced within the molecular system itself. Although quite preliminary at this time, research is already underway to construct artificial cells that are capable of self-replication (Szostak et al., 2001). Even further in the future and still rather speculative at this point are proposals for the construction of more general purpose molecular assemblers that are completely independent of existing biological systems (Drexler, 1992).

To clarify what is meant by a nanoscale programmable molecular assembler (NPMA), the following tentative definition is proposed: an NPMA is something of nanoscale dimensions that facilitates a direct, simple and unambiguous mapping between an

extensible and re-arrangeable sequence of symbols on the one hand, and a corresponding sequence of actions to assemble molecular components on the other. It is the ability for such control sequences to be extended and re-arranged sequences that confers the quality of programmability.

By this definition, it is asserted that ribosomes are currently the only practical nanoscale programmable molecular assemblers that are known to exist. There are other forms of template-based assembly, such as DNA and RNA polymerases. There are also non-template-based assembly systems such as self-splicing introns (Doudna and Cech, 2002b), but their products are inherently determined by the specific properties of their input sequences, and thus their assembly instructions are not truly symbolic. It is also possible that other more subtle biological assembly systems have been characterized in which sets of true symbols are mapped to sets of assembly actions. One example might be spliceosomes (Staley and Guthrie, 1998), but even here it has not been demonstrated that their catalytic assembly factors possess the above-mentioned properties of sequentiality and re-arrangeability to any significantly extensible degree that would still maintain a clear mapping to their assembly outputs.

At some point in the future it is possible that existing biomolecules may be discovered to possess these NPMA criteria, possibly in conjunction with powerful and previously unknown functionalities. However, at the current time, none of the other nanoscale molecular assembly systems currently known are truly comparable to the programmable capabilities of the ribosome. This generality is made possible by the use of an arbitrary code as defined by the set of tRNAs that link an anticodon at one end to a particular

amino acid at the other end. A set of enzymes is responsible for charging the tRNAs and the ribosome is responsible for using them. It is this distinct separation between the process that defines the meaning of the assembly instructions and the process that executes them that is ultimately responsible for the powerful symbolic capabilities of the protein translational system. With this extraordinary capability, the ribosome sits at the center of a process that is linked both to the past, by evolution from the RNA world, and to the future, by serving as a functional example for the creation of new nanoscale programmable molecular assemblers that may ultimately result in new forms of life.

References

1. Aho A, Sethi R, Ullman J. *Compilers, Principles, Techniques and Tools.* Addison-Wesley (1988).
2. Al Karadaghi S, Aevarsson A, Garber M, Zheltonosova J, Liljas A. *The structure of elongation factor G in complex with GDP: conformational flexibility and nucleotide exchange.* Structure 4:555-565 (1996).
3. Alberts J, Odell G. *In Silico Reconstitution of Listeria Propulsion Exhibits Nano-Saltation.* PLoS Biology 2:2054-2066 (2004).
4. Allen F, Almasi G, Andreoni W, Beece D, Berne B, Bright A, Brunheroto J, Cascaval C, Castanos J, Coteus P, Crumley P, Curioni A, Denneau M, Donath W, Eleftheriou M, Fitch B, Fleischer B, Georgiou C, Germain R, Giampapa M, Gresh D, Gupta M, Haring R, Ho H, Hochschild P, Hummel S, Jonas T, Lieber D, Martyna G, Maturu K, Moreira J, Newns D, Newton M, Philhower R, Picunko T, Pitera J, Pitman M, Rand R, Royyuru A, Salapura V, Sanomiya A, Shah R, Sham Y, Singh S, Snir M, Suits F, Swetz R, Swope W, Vishnumurthy N, Ward T, Warren H, Zhou R. *Blue Gene: A vision for protein science using a petaflop computer.* IBM SYSTEMS JOURNAL 40:310-327 (2001).
5. Allinger N, Miller M, VanCatledge F, Hirsch J. *Conformational Analysis. LVII. The Calculation of the Conformational Structures of Hydrocarbons by the Westheimer-Hendrickson-Wiberg Method.* J. Am. Chem. Soc. 89:4345-4357 (1967).
6. Allinger N. *Conformational Analysis. 130. MM2. A Hydrocarbon Force Field Utilizing V1 and V2 Torsional Terms.* J. Am. Chem. Soc. 99:8127-8134 (1977).
7. Allinger N, Chen K, Lii J. *An Improved Force Field (MM4) for Saturated Hydrocarbons.* J Comput Chem 17:642-668 (1996a).
8. Allinger N, Chen K, Katzenellenbogen J, Wilson S, Anstead G. *Hyperconjugative Effects on Carbon-Carbon Bond Lengths in Molecular Mechanics (MM4).* J Comput Chem 17:747-755 (1996b).
9. Allinger N, Durkin K. *Van der Waals Effects between Hydrogen and First-Row Atoms in Molecular Mechanics (MM3/MM4).* J Comput Chem 21:1229-1242 (2000).
10. Andreoni W, Curioni A, Mordasini T. *DFT-based molecular dynamics as a new tool for computational biology: First applications and perspective.* IBM J. RES. & DEV 45:397-407 (2001).
11. Angel E. *Interactive Computer Graphics, A top-down approach with OpenGL.* Addison-Wesley (1997).
12. Argaman N, Makov G. *Density functional theory: An introduction.* Am. J. Phys. 68:69-79 (2000).
13. Baker N, Sept D, Joseph S, Holst M, McCammon J. *Electrostatics of nanosystems: Application to microtubules and the ribosome.* PNAS 98:10037-10041 (2001).
14. Ban N, Nissen P, Hansen J, Moore P, Steitz T. *The Complete Atomic Structure of the Large Ribosomal Subunit at 2.4 A Resolution.* Science 289:905-920 (2000).
15. BannedByGaussian website. *Banned By Gaussian.* http://bannedbygaussian.org/ (2005).
16. Barber C, Dobkin D, Huhdanpaa H. *The Quickhull Algorithm for Convex Hulls.* ACM Trans. Math. Softw. 22:469-183 (1996).
17. Barden C, Schaefer H. *Quantum Chemistry in the 21st Century.* Pure Appl. Chem. 72:1405-1423 (2000).
18. Bashan A, Agmon I, Zarivach R, Schluenzen F, Harms J, Berisio R, Bartels H, Franceschi F, Auerbach T, Hansen H, Kossoy E, Kessler M, Yonath A. *Stuctural Basis of the Ribosomal Machinery for Peptide Bond Formation, Translocation, and Nascent Chain Progression.* Mol Cell 11:91-102 (2003).
19. Baumeister W, Steven A. *Macromolecular electron microscopy in the era of structural genomics.* Trends Biochem. Sci. 25:624-631 (2000).
20. Bayfield M, Dahlberg A, Schulmeister U, Dorner S, Barta A. *A conformational change in the ribosomal peptidyl transferase center upon active/inactive transition.* PNAS 98:10096-10101 (2001).

21. Bayly C, Cieplak P, Cornell W, Kollman P. *A Well-Behaved Electrostatic Potential Based Method Using Charge Restraints for Deriving Atomic Charges: The RESP Model.* J. Phys. Chem. 97:10269-10280 (1993).
22. Benkler Y. *Commons-Based Strategies and the Problems of Patents.* Science 305:1110-1111 (2004).
23. Berman H, Westbrook J, Feng Z, Gilliland G, Bhat T, Weissig H, Shindyalov I, Bourne P. *The Protein Data Bank.* Nucleic Acids Research 28:235-242 (2000).
24. Binkley J, Pople J, Hehre W. *Self-Consistent Molecular Orbital Methods. 21. Small Split-Valence Basis Sets for First-Row Elements.* J. Am. Chem. Soc. 102:939-947 (1980).
25. Biou V, Cherfils J. *Structural Principles for the Multispecificity of Small GTP-Binding Proteins.* Biochemistry 43:6833-6840 (2004).
26. Blackburn G, Gait M. *Nucleic Acids in Chemistry and Biology.* Oxford University Press (1996).
27. Blanchard S, Puglisi J. *Solution structure of the A loop of 23S ribosomal RNA.* PNAS 98:3720-3725 (2001).
28. Blanchard S, Gonzalez R, Kim H, Chu S, Puglisi J. *tRNA selection and kinetic proofreading in translation.* Nature Structural Biology 11:1008-1014 (2004).
29. Bourg D. *Physics for Game Developers.* O'Reilly (2002).
30. Bowen W, Hill W, Lodmell J. *Comparison of rRNA Cleavage by Complementary 1,10-Phenanthroline-Cu(II)- and EDTA-Fe(II)-Derivatized Oligonucleotides.* Methods 25:344-350 (2001).
31. Brunger A, Adams P, Rice L. *Recent developments for the efficient crystallographic refinement of macromolecular structure.* Curr Opin Struct Biol 8:606-611 (1998).
32. Bsm. *Het PDB groups listing, Biomolecular Structure and Modeling, University College London.* http://www.biochem.ucl.ac.uk/bsm/pdbsum/hetgroups/ (2004).
33. Burkhardt C, Zacharias M. *Modeling ion binding to AA platform motifs in RNA: a continuum solvent study including conformational adaptation.* Nucleic Acids Research 29:3910-3918 (2001).
34. Butler L. *Chemical Reaction Dynamics Beyond the Born-Oppenheimer Approximation.* Annu. Rev. Phys. Chem. 49:125-171 (1998).
35. CB website, Raymond E. *The Cathedral and the Bazaar.* http://www.catb.org/~esr/writings/cathedral-bazaar/ (2005).
36. CHIME website. *MDL CHIME.* http://www.mdl.com/products/framework/chime/ (2005).
37. CRW website. *Gutell Lab Comparative RNA Web Site.* http://www.rna.icmb.utexas.edu/ (2005).
38. Capelle K. *A bird's-eye view of density-functional theory.* Cond. Matter 0211443:1-27 (2003).
39. Car R, Parrinello M. *Unified Approach for Molecular Dynamics and Density-Functional Theory.* Phys Rev Lett. 55:2471-2474 (1985).
40. Carey R, Bell G. *The Annotated VRML 2.0 Reference Manual.* Addison-Wesley (1997).
41. Carter A, Clemons W, Brodersen D, Morgan-Warren R, Wimberly B, Ramakrishnan V. *Functional insights from the structure of the 30S ribosomal subunit and its interactions with antibiotics.* Nature 407:340-348 (2000).
42. Carter Andrew, Clemons William, Brodersen Ditlev, Morgan-Warren Robert, Hartsch Thomas, Wimberly Brian, Ramakrishnan V. *Crystal Structure of an Initiation Factor Bound to the 30S Ribosomal Subunit.* Science 291:498-501 (2001).
43. Cate J, Yusupov M, Yusupova G, Earnest T, Noller H. *X-ray Crystal Structures of 70S Ribosome Functional Complexes.* Science 285:2095-2104 (1999).
44. Cates M, Teodoro M, Phillips G. *Molecular Mechanisms of Calcium and Magnesium Binding to Parvalbumin.* Biophysical Journal 82:1133-1146 (2002).
45. Chayen N. *Turning protein crystallisation from an art into a science.* Curr Opin Struct Biol 14:577-583 (2004).
46. Cheatham T, Miller J, Fox T, Darden T, Kollman P. *Molecular Dynamics Simulations on Solvated Biomolecular Systems: The Particle Mesh Ewald Method Leads to Stable Trajectories of DNA, RNA, and Proteins.* J. Am. Chem. Soc. 117:4193-4194 (1995).

47. Cheatham T, Kollman P. *Observation of the A-DNA to B-DNA Transition During Unrestrained Molecular Dynamics in Aqueous Solution.* J. Mol. Biol. 259:434-444 (1996).
48. Cheatham T, Kollman P. *Molecular Dynamics Simulations Highlight the Structural Differences among DNA:DNA, RNA:RNA, and DNA:RNA Hybrid Duplexes.* J. Am. Chem. Soc. 119:4805-4825 (1997).
49. Cheatham T. *Simulation and modeling of nucleic acid structure, dynamics and interactions.* Curr Opin Struct Biol 14:360-367 (2004).
50. Clemons W, Brodersen D, McCutcheon J, May J, Carter A, Morgan Warren R, Wimberly B, Ramakrishnan V. *Crystal Structure of the 30S Ribosomal Subunit from Thermus thermophilus: Purification, Crystallization and Structure Determination.* J. Mol. Biol. 310:827-843 (2001).
51. Cn3D website. *Cn3D Home Page.* http://www.ncbi.nlm.nih.gov/Structure/CN3D/cn3d.shtml (2005).
52. Cornell W, Cieplak P, Bayly C, Gould I, Merz K, Ferguson D, Spellmeyer D, Fox T, Caldwell J, Kollman P. *A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules.* J. Am. Chem. Soc. 117:5179-5197 (1995).
53. Cramer C, Truhlar D. *Implicit Solvation Models: Equlibria, Structure, Spectra, and Dynamics.* Chem. Rev. 99:2161-2200 (1999).
54. Crick F. *What Mad Pursuit. A Personal View of Scientific Discovery.* Basic Books (1988).
55. Culver Gloria. *Meanderings of the mRNA through the Ribosome.* Structure 9:751-758 (2001).
56. Daniel R, Dunn R, Finney J, Smith J. *The Role of Dynamics in Enzyme Activity.* Annu. Rev. Biophys. Biomol. Struct. 32:69-92 (2003).
57. Darden T, Perera L, Li L, Pedersen L. *New tricks for modelers from the crystallography toolkit: the particle mesh Ewald algorithm and its use in nucleic acid simulations.* Structure 7:R55-R60 (1999).
58. Darnell P, Margolis P. *C. A Software Engineering Approach.* Springer-Verlag (1991).
59. DeLano W. *The PyMOL Molecular Graphics Systems.* http://www.pymol.org (2002).
60. DeLano W. *The case for open-source software in drug discovery.* Drug Discovery Today 10:213-217 (2005).
61. Doudna Jennifer, Rath Virginia. *Structure and Function of the Eukaryotic Ribosome: The Next Frontier.* Cell Vol. 109:153-156 (2002a).
62. Doudna J, Cech T. *The chemical repertoire of natural ribozymes.* Nature 418:222-228 (2002b).
63. Drexler K. *Nanosystems, Molecular Machinery, Manufacturing and Computation.* Wiley (1992).
64. Duan Y, Kollman P. *Pathways to a Protein Folding Intermediate Observed in a 1-Microsecond Simulation in Aqueous Solution.* Science 282:740-744 (1998).
65. Duan Y, Wu C, Chowdhury S, Lee M, Xiong G, Zhang W, Yang R, Cieplak P, Luo R, Lee T, Caldwell J, Wang J, Kollman P. *A Point-Charge Force Field for Molecular Mechanics Simulations of Proteins Based on Condensed-Phase Quantum Mechanical Calculations.* J Comput Chem 24:1999-2012 (2003).
66. Duarte C, Pyle A. *Stepping Through an RNA Structure: A Novel Approach to Conformational Analysis.* J. Mol. Biol. 284:1465-1478 (1998).
67. Ehmann S, Lin M. *Accurate and Fast Proximity Queris Between Polyhedra Using Convex Surface Decomposition.* EUROGRAPHICS 20:C500-C510 (2001).
68. Em data website. *The Macromolecular Structure Database, EMBL.* http://www.ebi.ac.uk/msd/ (2005).
69. Ermer O, Lifson S. *Consistent Force Field Calculations. III. Vibrations, Conformations, and Heats of Hydrogenation of Nonconjugated Olefins.* J. Am. Chem. Soc. 95:4121-4132 (1973).
70. Fersht A, Daggett V. *Protein Folding and Unfolding at Atomic Resolution.* Cell 108:573-582 (2002).
71. Fogolari F, Brigo A, Molinari H. *The Poisson-Boltzmann equation for biomolecular electrostatics: a tool for structural biology.* J. Mol. Recognit. 15:377-392 (2002).
72. Folding website. *Folding@home distributed computing website.* http://folding.stanford.edu/ (2005).
73. Foloppe N, MacKerell A. *All-Atom Empirical Force Field for Nucleic Acids: I. Parameter Optimization Based on Small Molecule and Condensed Phase Macromolecular Target Data.* J Comput Chem 21:86-104 (2000).

74. Foote B, Yoder J. *Big Ball of Mud.* http://www.laputan.org/mud/mud.html (1999).
75. Forester T, Smith W. *SHAKE, Rattle, and Roll: Efficient Constraint Algorithms for Linked Rigid Bodies.* J Comput Chem 19:102-111 (1998).
76. Foster I. *Designing and Building Parallel Programs.* Addison-Wesley (1995).
77. Frank J, Agrawal R. *A ratchet-like inter-subunit reorganization of the ribosome during translocation.* Nature 406:318-322 (2000).
78. Frank J. *Electron Microscopy of Functional Ribosome Complexes.* Biopolymers Vol 68:223-233 (2003).
79. Frank J, Sengupta J, Gao H, Li W, Valle M, Zavialov A, Ehrenberg M. *The role of tRNA as a molecular spring in decoding, accomodation, and peptidyl transfer.* FEBS Letters 579:959-562 (2005).
80. Fuller S. *Depositing Electron Microscopy Maps.* Structure 11:11-12 (2003).
81. GNU website, Stallman R. *The GNU Operating System.* http://www.gnu.org/ (2005).
82. Gajewski J, Gilbert K, Kreek T. *General Molecular Mechanics Approach to Transition Metal Complexes.* J Comput Chem 19:1167-1178 (1998).
83. Gao M, Wilmanns M, Schulten K. *Steered Molecular Dynamics Studies of Titin I1 Domain Unfolding.* Biophysical Journal 83:3435-3445 (2002).
84. Garrett R, Grisham C. *Biochemistry.* Saunders College Publishing (1999).
85. Garrett R, Douthwaite S, Liljas A, Matheson A, Moore P, Noller H. *The Ribosome. Structure, Function, Antibiotics, and Cellular Interactions.* ASM Press (2000).
86. Gibrat J, Madej T, Bryant S. *Surprising similarities in structure comparison.* Curr Opin Struct Biol 6:377-385 (1996).
87. Giles J. *Software company bans competitive users.* Nature 429:231 (2004).
88. Gilson M. *Theory of electrostatic interactions in macromolecules.* Curr Opin Struct Biol 5:216-223 (1995).
89. Goodman L, Pophristic V, Weinhold F. *Origin of Methyl Internal Rotation Barriers.* Acc. Chem. Res. 32:983-993 (1999).
90. Grayson P, Tajkhorshid E, Schulten K. *Mechanisms of Selectivity in Channels and Enzymes Studied with Interactive Molecular Dynamics.* Biophysical Journal 85:36-48 (2003).
91. Green R, Lorsch J. *The Path to Perdition is Paved with Protons.* Cell 110:665-668 (2002).
92. Gruber D. *The Mathematics of the 3D Rotation Matrix.* http://www.makegames.com/3drotation/ (2000).
93. Gualerzi C, Pon C. *Initiation of mRNA Translation in Prokaryotes.* Biochemistry 29:5881-5889 (1990).
94. Guelich S, Gundavaram S, Birznieks G. *CGI Programming with Perl.* O'Reilly (2000).
95. Guerra C, Bickelhaupt F, Snijders J, Baerends E. *Hydrogen Bonding in DNA Base Pairs: Reconciliation of Theory and Experiment.* J. Am. Chem. Soc. 122:4117-4128 (2000).
96. Gutell R, Lee J, Cannone J. *The accuracy of ribosomal RNA comparative structure models.* Curr Opin Struct Biol 12:301-310 (2002).
97. Halgren T. *Merck Molecular Force Field. I. Basis, Form, Scope, Parametrization, and Performance of MMFF94.* J Comput Chem 17:490-519 (1996a).
98. Halgren T. *Merck Molecular Force Field. II. MMFF94 van der Waals and Electrostatic Parameters for Intermolecular Interactions.* J Comput Chem 17:520-552 (1996b).
99. Halgren T. *Merck Molecular Force Field. III. Molecular Geometries and Vibrational Frequencies for MMFF94.* J Comput Chem 17:553-586 (1996c).
100. Halgren T. *MMFF VII. Characterization of MMFF94, MMFF94s, and Other Widely Available Force Fields for Conformational Energies and for Intermolecular-Interaction Energies and Geometries.* J Comput Chem 20:730-748 (1999a).
101. Halperin I, Ma B, Wolfson H, Nussinov R. *Principles of Docking: An Overview of Search Algorithms and a Guide to Scoring Functions.* Proteins 47:409-443 (2002).

102. Hansen J, Schmeing T, Moore P, Steitz T. *Structural insights into peptide bond formation.* PNAS 99:11670-11675 (2002).
103. Harrison S. *Whither structural biology?.* Nature Structural Biology 11:12-15 (2004).
104. Harvey S, Wang C, Teletchea S, Lavery R. *Motifs in Nucleic Acids: Molecular Mechanics Restraints for Base Pairing and Base Stacking.* J Comput Chem 24:1-9 (2003).
105. Hennelly S, Antoun A, Ehrenberg M, Gualerzi C, Knight W, Lodmell J, Hill W. *A Time-resolved Investigation of Ribosomal Subunit Association.* J. Mol. Biol. 346:1243-1258 (2005).
106. Hershkovitz E, Tannenbaum E, Howerton S, Sheth A, Tannenbaum A, Williams L. *Automated identification of RNA conformational motifs: theory and application to the HM LSU 23S rRNA.* Nucleic Acids Research 31:6249-6257 (2003).
107. Hill W, Dahlberg A, Garrett R, Moore P, Schlessinger D, Warner J. *The Ribosome: Structure, Function & Evolution.* American Society for Microbiology (1990).
108. Hinsen K. *The Molecular Modeling Toolkit: A New Approach to Molecular Simulations.* J Comput Chem Vol. 21, No. 2:79-85 (2000).
109. Hohenberg P, Kohn W. *Inhomogeneous Electron Gas.* Phys. Rev. 136:B864-B871 (1964).
110. Holm L, Sander C. *Searching Protein Structure Databases Has Come of Age.* Proteins 19:165-173 (1994).
111. Hudson T, Lin M, Cohen J, Gottschalk S, Manocha D. *V-COLLIDE: Accelerated Collision Detection for VRML.* Proc. VRML'97 1:1-7 (1997).
112. Humphrey W, Dalke A, Schulten K. *VMD - Visual Molecular Dynamics.* J Molec Graphics 14:33-38 (1996).
113. Hunter P, Borg T. *Integration from proteins to organs: the Physiome Project.* Nat Rev Mol Cell Biol 4:237-243 (2003).
114. Hunter P. *The IUPS Physiome Project: a framework for computational physiology.* Prog Biophys Mol Biol. 85:551-569 (2004).
115. Hutter J, Carloni P, Parrinello M. *Nonempirical Calculations of a Hydrated RNA Duplex.* J. Am. Chem. Soc. 118:8710-8712 (1996).
116. Ierusalimschy R, de Figueiredo L, Celes W. *Lua - an extensible extension language.* Software Practice and Experience 26:635-652 (1996).
117. Isralewitz B, Gao M, Schulten K. *Steered molecular dynamics and mechanical functions of proteins.* Curr Opin Struct Biol 11:224-230 (2001a).
118. Isralewitz B, Baudry J, Gullingsrud J, Kosztin D, Schulten K. *Steered molecular dynamics investigations of protein function.* J Mol Graph Model. 19:13-25 (2001b).
119. Izrailev S, Stepaniants S, Balsera M, Oono Y, Schulten K. *Molecular dynamics study of unbinding of the avidin-biotin complex.* Biophysical Journal 72:1568-1581 (1997).
120. Izrailev S, Stepaniants S, Isralewitz B, Kosztin D, Lu H, Molnar F, Wriggers W, Schulten K. *Steered Molecular Dynamics, in Computational Molecular Dynamics: Challenges, Methods, Ideas, Vol. 4.* Springer-Verlag (1998).
121. Jamsa K, Cope K. *Internet Programming.* Jamsa Press (1995).
122. Jelsch C, Teeter M, Lamzin V, Pichon-Pesme V, Blessig R, Lecomte C. *Accurate protein crystallography at ultra-high resolution: Valence electron distribution in crambin.* PNAS 97:3171-3176 (2000).
123. Jmol website, Gezelter D, Smith B, Willighagen E, Howard M. *Jmol.* http://jmol.sourceforge.net/ (2005).
124. Jones S, Daley D, Luscombe N, Berman H, Thornton J. *Protein-RNA interactions: a structural analysis.* Nucleic Acids Research 29:943-954 (2001).
125. Jorgensen W, Madura J, Swenson C. *Optimized Intermolecular Potential Functions for Liquid Hydrocarbons.* J. Am. Chem. Soc. 106:6638-6646 (1984).

126. Jorgensen W, Maxwell D, Tirado Rives J. *Development and Testing of the OPLS All-Atom Force Field on Conformational Energetics and Properties of Organic Liquids.* J. Am. Chem. Soc. 118:11225-11236 (1996).
127. Joyce G. *The antiquity of RNA-based evolution.* Nature 418:214-221 (2002).
128. Kale L, Skeel R, Bhandarkar M, Brunner R, Gursoy A, Krawetz N, Phillips J, Shinozaki A, Varadarajan K, Schulten K. *NAMD2: Greater scalability for parallel molecular dynamics.* J. Comp. Phys. 151:283-312 (1999).
129. Karplus M, McCammon J. *Molecular dynamics simulations of biomolecules.* Nature Structural Biology 9:646-652 (2002).
130. Katunin V, Muth G, Strobel S, Wintermeyer W, Rodnina M. *Important Contribution to Catalysis of Peptide Bond Formation by a Single Ionizing Group within the Ribosome.* Mol Cell 10:339-346 (2002).
131. Kauffman S. *The Origins of Order.* Oxford University Press (1993).
132. Klaholz B, Pape T, Zavialov A, Myasnikov A, Orlova E, Vestergaard B, Ehrenberg M, van Heel M. *Structure of the Escherichia coli ribosomal termination complex with release factor 2.* Nature 421:90-94 (2003).
133. Kohn W, Sham L. *Self-Consistent Equations Including Exchange and Correlation Effects.* Phys. Rev. 140:A1133-A1138 (1965).
134. Kohn W. *Nobel Lecture: Electronic structure of matter-wave functions and density functionals.* Rev. Mod. Phys. 71:1253-1266 (1999).
135. Kollman P, Massova I, Reyes C, Kuhn B, Huo S, Chong L, Duan Y, Wang W, Donini O, Cieplak P, Srinivasan J, Case D, Cheatham T. *Calculating Structures and Free Energies of Complex Molecules: Combining Molecular Mechanics and Continuum Models.* Acc. Chem. Res. 33:889-897 (2000).
136. Krebs W, Gerstein M. *The morph server: a standardized system for analyzing and visualizing macromolecular motions in a database framework.*. Nucleic Acids Research 28:1665-1675 (2000).
137. Kruger P, Verheyden S, Declerck P, Engelborghs Y. *Extending the capabilities of targeted molecular dynamics: Simulation of a large conformational transition in plasminogen activator inhibitor 1.* Protein Science 10:798-808 (2001).
138. Lata K, Agrawal R, Penczek P, Grassucci R, Zhu J, Frank J. *Three-dimensional Reconstruction of the Escherichia coli 30 S Ribosoma Subunit in Ice.* J. Mol. Biol. 262:43-52 (1996).
139. Laurberg M, Kristensen O, Martemyanov K, Gudkov A, Nagaev I, Hughes D, Liljas A. *Structure of a Mutant EF-G Reveals Domain III and Possibly the Fusidic Acid Binding Site.* J. Mol. Biol. 303:593-603 (2000).
140. Lee G, Nowak W, Jaroniec J, Zhang Q, Marszalek P. *Molecular Dynamics Simulations of Forced Conformational Transitions in 1,6-Linked Polysaccharides.* Biophysical Journal 87:1456-1465 (2004).
141. Levitt M. *The birth of computational structural biology.* Nature Structural Biology 8:392-393 (2001).
142. Liao S, Seeman N. *Translation of DNA Signals into Polymer Assembly Instructions.* Science 306:2072-2074 (2004).
143. Lim V, Curran J. *Analysis of codon:anticodon interactions within the ribosome provides new insights into codon reading and the genetic code structure.* RNA 7:942-957 (2001).
144. Lindahl E, Hess B, van der Spoel D. *GROMACS 3.0: a package for molecular simulation and trajectory analysis.* J Mol Model 7:306-317 (2001).
145. Linux Magazine. *Open Source 3D Package for Game Creation, Crystal Space.* http://www.linux-mag.com/2003-02/potm_01.html (2003).
146. Lodmell J, Dahlberg A. *A Conformational Switch in Escherichia coli 16S Ribosomal RNA During Decoding of Messenger RNA.* Science 277:1262-1267 (1997).
147. Lolle S, Victor J, Young J, Pruitt R. *Genome-wide non-mendelian inheritance of extra-genomic information in Arabidopsis.* Nature 434:505-509 (2005).
148. Lorensen W, Cline H. *Marching Cubes: A High Resolution 3D Surface Construction Algorithm.* Computer Graphics 21:163-169 (1987).

149. Lovell S, Word J, Richardson J, Richardson D. *The Penultimate Rotamer Library.* Proteins 40:389-408 (2000).
150. Lutz M. *Programming Python.* O'Reilly (2001).
151. Ma B, Lii J, Allinger N. *Molecular Polarizabilities and Induced Dipole Moments in Molecular Mechanics.* J Comput Chem 21:813-825 (2000).
152. MacKerell A, Wiorkiewicz-Kuczera J, Karplus M. *An All-Atom Empirical Energy Function for the Simulation of Nucleic Acids.* J. Am. Chem. Soc. 117:11946-11975 (1995).
153. MacKerell A. *Influence of Magnesium Ions on Duplex DNA Structural, Dynamic, and Solvation Properties.* J. Phys. Chem. B 101:646-650 (1997).
154. MacKerell A, Bashford D, Bellot M, Dunbrack R, Evansec J, Field M, Fischer S, Gao J, Guo H, Ha S, Joseph McCarthy D, Kuchnir L, Kuczera K, Lau F, Mattos C, Michnick S, Ngo T, Nguyen D, Prodhom B, Reiher W, Roux B, Schlenkrich M, Smith J, Stote R, Straub J, Watanabe M, Wiorkiewicz Kuczera J, Yin D, Karplus M. *All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins.* J. Phys. Chem. 102:3586-3616 (1998).
155. MacKerell A, Banavali N. *All-Atom Empirical Force Field for Nucleic Acids: II. Application to Molecular Dynamics Simulations of DNA and RNA in Solution.* J Comput Chem 21:105-120 (2000).
156. MacKerell A. *Empirical Force Fields for Biological Macromolecules: Overview and Issues.* J Comput Chem 25:1584-1604 (2004a).
157. Macke T, Svrcek-Seiler W, Case D. *Nucleic Acid Builder Users' Manual.* http://www.scripps.edu/case/casegr-sh-3.2.html (2004).
158. Maden B. *Historical review: Peptidyl transfer, the Monro era.* Trends Biochem. Sci. 28:619-624 (2003).
159. Martz E. *Protein Explorer: easy yet powerful macromolecular visualization.* Trends Biochem. Sci. 27:107-109 (2002).
160. Marzi S, Knight W, Brandi L, Caserta E, Soboleva N, Hill W, Gualerzi C, Lodmell J. *Ribosomal localization of translation initiation factor IF2.* RNA 9:958-969 (2003).
161. Medalia O, Weber I, Frangakis A, Nicastro D, Gerisch G, Baumeister W. *Macromolecular Architecture in Eukaryotic Cells Visualized by Cryoelectron Tomography.* Science 298:1209-1213 (2002).
162. Melchionna S. *Molecular Dynamics simulation of biosystems: Perspectives and open problems.* Mem. S.A.It. Suppl. 4:75-81 (2004).
163. Mingos D, Lyne P, Pidun U. *Recent developments in the application of theory to structural problems in coordination chemistry.* Pure Appl. Chem. 67:265-272 (1995).
164. Miyazawa A, Fujiyoshi Y, Unwin N. *Structure and gating mechanism of the acetylcholine receptor pore.* Nature 423:949-955 (2003).
165. Mohr D, Wintermeyer W, Rodnina M. *GTPase Activation of Elongation Factors Tu and G on the Ribosome.* Biochemistry 41:12520-12528 (2002).
166. Moll I, Hirokawa G, Kiel M, Kaji A, Blasi U. *Translation initiation with 70S ribosomes: an alternative pathway for leaderless mRNAs.* Nucleic Acids Research 32:3354-3363 (2004).
167. Moller T, Haines E. *Real-Time Rendering.* A K Peters, Ltd. (1999).
168. Momany F. *Determination of Partial Atomic Charges from Ab Initio Molecular Electrostatic Potentials. Application to Formamide, Methanol, and Formic Acid.* J. Phys. Chem. 82:592-601 (1978).
169. Moody G. *Rebel Code. The inside story of Linux and the Open Source Revolution.* Perseus Publishing (2001).
170. Moore P. *Structural Motifs in RNA.* Annu. Rev. Biochem. 68:287-300 (1999).
171. Moore P, Steitz T. *After the ribosome structures: How does peptidyl transferase work?.* RNA 9:155-159 (2003b).
172. Morokuma K. *New challenges in quantum chemistry: quests for accurate calculations for large molecular systems.* Phil. Trans. R. Soc. Lond. 360:1149-1164 (2002).

173. Murray L, Arendall W, Richardson D, Richardson J. *RNA backbone is rotameric.* PNAS 100:13904-13909 (2003).
174. Murthy V, Srinivasan R, Draper D, Rose G. *A Complete Conformational Map for RNA.* J. Mol. Biol. 291:313-327 (1999).
175. Muth Gregory, Ortoleva-Donnelly Lori, Strobel Scott. *A Single Adenosine with a Neutral pKa in the Ribosomal Transferase Center.* Science 289:947-950 (2000).
176. Muth G, Chen L, Kosek A, Strobel S. *pH-dependent conformational flexibility within the ribosomal peptidyl transferase center.* RNA 7:1403-1415 (2001).
177. Nevins N, Allinger N. *Molecular Mechanics (MM4) Vibrational Frequency Calculations for Alkenes and Conjugated Hydrocarbons.* J Comput Chem 17:730-746 (1996c).
178. Nissen P, Kjeldgaard M, Thirup S, Polekhina G, Reshetnikova L, Clark B, Nyborg J. *Crystal structure of the ternary complex of Phe-tRNAPhe, EF-Tu, and a GTP analog.* Science 270:1464-1469 (1995).
179. Nissen P, Hansen J, Ban N, Moore P, Steitz T. *The Structural Basis of Ribosome Activity in Peptide Bond Synthesis.* Science 289:920-930 (2000).
180. Ogle J, Brodersen D, Clemons W, Tarry M, Carter A, Ramakrishnan V. *Recognition of Cognate Transfer RNA by the 30S Ribosomal Subunit.* Science 292:897-902 (2001).
181. Ogle J, Murphy F, Tarry M, Ramakrishnan V. *Selection of tRNA by the Ribosome Requires a Transition from an Open to a Closed Form.* Cell 111:721-723 (2002).
182. Ogle J, Carter A, Ramakrishnan V. *Insights into the decoding mechanism from recent ribosome structures.* Trends Biochem. Sci. 28:259-266 (2003).
183. Oldfield E. *Chemical Shifts in Amino Acids, Peptides, and Proteins: From Quantum Chemistry to Drug Design.* Annu. Rev. Phys. Chem. 53:349-378 (2002).
184. Ousterhout J. *Scripting: Higher Level Programming for the 21st Century.* IEEE Computer 31:23-30 (1998).
185. Pande V, Baker I, Chapman J, Elmer S, Khaliq S, Larson S, Rhee Y, Shirts M, Snow C, Sorin E, Zagrovic B. *Atomistic Protein Folding Simulations on the Submillisecond Time Scale Using Worldwide Distributed Computing* . Biopolymers 68:91-109 (2003).
186. Parisi T. *Like a Phoenix From the Ashes: X3D and the Rebirth of Reason.* http://flux.typepad.com/the_flux_papers/2004/08/project_phoenix.html (2004).
187. PdbBeta site. *PDB Current Holdings Breakdown.* http://pdbbeta.rcsb.org/pdb/holdings.do (2005).
188. Pioletti M, Schlunzen F, Harms J, Zarivach R, Bashan A, Bartels H, Auerbach T, Jacobi C, Hartsch T, Yonath A, Franceschi F. *Crystal structures of complexes of the small ribosomal subunit with tetracycline, edeine and IF3.* EMBO Journal 20:1829-1839 (2001).
189. Poole A, Jeffares D, Penny D. *The Path from the RNA World.* J Mol Evol 46:1-17 (1998).
190. Pople J. *Quantum Chemical Models.* http://nobelprize.org/chemistry/laureates/1998/pople-lecture.html (1998).
191. Povray Org. *The Persistence of Vision Raytracer.* http://www.povray.org (2005).
192. Pritchard H. *Computational Chemistry in the 1950s.* J Mol Graph Model. 19:623-627 (2001).
193. Pulay P, Fogarasi G, Pang F, Boggs J. *Systematic ab Initio Gradient Calculation of Molecular Geometries, Force Constants, and Dipole Moment Derivatives.* J. Am. Chem. Soc. 101:2550-2560 (1979).
194. RCSB. *PDB Format Description Version 2.2.* http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2_frame.html (2004).
195. Ramakrishnan V. *Ribosome Structure and the Mechanism of Translation.* Cell 108:557-572 (2002).
196. Ray E. *Learning XML.* O'Reilly (2001).
197. Robins N. *The OpenGL Utility Toolkit (GLUT).* http://www.xmission.com/~nate/glut.html (2001).
198. Rodnina M, Wintermeyer W. *Ribosome fidelity: tRNA discrimintation, proofreading and induced fit.* Trends Biochem. Sci. 26:124-130 (2001).
199. Rodnina M, Wintermeyer W. *Peptide bond formation on the ribosome: structure and mechanism.* Curr Opin Struct Biol 13:334-340 (2003).

200. Rodnina M, Gromadski K, Kothe U, Wieden H. *Recognition and selection of tRNA in translation.* FEBS Letters 579:938-942 (2005).
201. Sanbonmatsu K, Joseph S. *Understanding Discrimination by the Ribosome: Stability Testing and Groove Measurement of Codon-Anticodon Pairs.* J. Mol. Biol. 328:33-47 (2003).
202. Savchenko S. *3D Graphics Programming, Games and Beyond.* Sams Publishing (2000).
203. Sayle R, Milner-White E. *RASMOL: biomolecular graphics for all.* Trends Biochem. Sci. 20:374-376 (1995).
204. Schlecht M. *Molecular Modeling on the PC.* John Wiley and Sons (1998).
205. Schlegel H. *Exploring Potential Engery Surfaces for Chemical Reactions: An Overview of Some Practical Methods.* J Comput Chem 24:1514-1527 (2003).
206. Schlick T, Barth E, Mandziuk M. *Biomolecular Dynamics at Long Timesteps: Bridging the Timescale Gap Between Simulation and Experimentation.* Annu. Rev. Biophys. Biomol. Struct. 26:181-222 (1997).
207. Schlick T. *Molecular Modeling and Simulation.* Springer (2002).
208. Schlitter J, Swegat W, Mulders T. *Distance-type reaction coordinates for modeling activated processes.* J Mol Model 7:171-177 (2001).
209. Schluenzen F, Tocilj A, Zarivach R, Harms J, Gluehmann M, Janell D, Bashan A, Bartels H, Agmon I, Franceschi F, Yonath A. *Structure of Functionally Activated Small Ribosomal Subunit at 3.3 A Resolution.* Cell 102:615-623 (2000).
210. Schmitt E, Panvert M, Blanquet S, Mechulam Y. *Crystal structure of methionyl-tRNAfMet transformylase complexed with the initiator formyl-methionyl-tRNAfMet.* EMBO Journal 17:6819-6826 (1998).
211. Schreiber H, Steinhauser O. *Cutoff Size Does Strongly Influence Molecular Dynamics Results on Solvated Polypeptides.* Biochemistry 31:5856-5860 (1992).
212. Schrodinger E. *The fundamental idea of wave mechanics.* http://nobelprize.org/physics/laureates/1933/schrodinger-lecture.html (1933).
213. Scott W, Hunenberger P, Tironi I, Mark A, Billeter S, Fennen J, Torda A, Huber T, Kruger P, van Gunsteren W. *The GROMOS Biomolecular Simulation Program Package.* J. Phys. Chem. 103:3596-3607 (1999).
214. Shindyalov I, Bourne P. *Protein structure alignment by incremental combinatorial extension (CE) of the optimal path.* Protein Engineering 11:739-747 (1998).
215. Slepchenko B, Schaff J, Macara I, Loew L. *Quantitative cell biology with the Virtual Cell.* Trends Biochem. Sci. 13:570-576 (2003).
216. Sorin E, Engelhardt M, Herschlag D. *RNA Simulations: Probing Hairping Unfolding and the Dynamics of a GNRA Tetraloop.* J. Mol. Biol. 317:493-506 (2002).
217. Sorin E, Rhee Y, Nakatani B, Pande V. *Insights into Nucleic Acid Conformational Dynamics from Massively Parallel Stochastic Simulations.* Biophysical Journal 85:790-803 (2003).
218. Sorin E, Rhee Y, Pande V. *Does water play a structural role in the folding of small nucleic acids?.* Biophysical Journal epub:1-28 (2005).
219. Southworth D, Green R. *Ribosomal Translocation: Sparsomycin Pushes the Button.* Current Biology 13:652-654 (2003).
220. Spahn C, Beckmann R, Eswar N, Penczek P, Sali A, Blobel G, Frank J. *Structure of the 80S Ribosome from Saccharomyces cerevisiae -- tRNA-Ribosome and Subunit-Subunit Interactions.* Cell 107:373-386 (2001).
221. Spolsky J. *The Schlemiel way of software.* http://www.salon.com/tech/feature/2004/12/09/spolsky/ (2004).
222. Sprik M, Hutter J, Parrinello M. *Ab initio molecular dynamics simulation of liquid water: Comparison of three gradient-corrected density functionals.* J. Chem. Phys. 105:1142-1152 (1996).
223. Staley J, Guthrie C. *Mechanical Devices of the Spliceosome: Motors, Clocks, Springs, and Things.* Cell 92:315-326 (1998).

224. Standish T. *Data Structures in Java.* Addison-Wesley (1997).
225. Stark H, Rodnina M, Wieden H, Zemlin F, Wintermeyer W, van Heel M. *Ribosome interactions of aminoacyl-tRNA and elongation factor Tu in the codon-recognition complex.* Nature Structural Biology 9:849-854 (2002).
226. Steitz T. *On the structural basis of peptide-bond formation and antibiotic resistance from atomic structures of the large ribosomal subunit.* FEBS Letters 579:955-958 (2005).
227. Stroustrup B. *The C++ Programming Language, 3rd Ed.*. Addison-Wesley (1997).
228. Szabo A, Ostlund N. *Modern Quantum Chemistry.* Dover (1989).
229. Szostak J, Bartel D, Luisi P. *Synthesizing life.* Nature 409:387-390 (2001).
230. Takyar S, Hickerson R, Noller H. *mRNA Helicase activity of the Ribosome.* Cell 120:49-58 (2005).
231. Teragrid website. *About the TeraGrid.* http://www.teragrid.org/about/index.html (2005).
232. Thompson J, Higgins D, Gibson T. *CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice.* Nucleic Acids Research 22:4673-4680 (1994).
233. Tomasi J. *Thirty years of continuum solvation chemistry: a review, and prospects for the near future.* Theor Chem Acc 112:184-203 (2004).
234. Trolltech. *Qt Reference Documentation.* http://doc.trolltech.com/2.3/index.html (2004).
235. Tse J. *Ab Initio Molecular Dynamics with Density Functional Theory.* Annu. Rev. Phys. Chem. 53:249-90 (2002).
236. Tuerk C, Gold L. *Systematic evolution of ligands by exponential enrichment: RNA ligands to bacteriophage T4 DNA polymerase* . Science 249:505-511 (1990).
237. Valle M, Zavialov A, Sengupta J, Rawat U, Ehrenberg M, Frank J. *Locking and Unlocking of Ribosomal Motions.* Cell 114:123-134 (2003a).
238. Valle M, Zavialov A, Li W, Stagg S, Sengupta J, Nielsen R, Nissen P, Harvey S, Ehrenberg M, Frank J. *Incorporation of aminoacyl-tRNA into the ribosome as seen by cryo-electron microscopy.* Nature Structural Biology 10:899-1074 (2003b).
239. Van Alsenoy C, Yu C, Peeters A, Martin J, Schafer L. *Ab Initio Geometry Determination of Proteins. 1. Crambin.* J. Phys. Chem. 102:2246-2251 (1998).
240. Van Houten J. *A Century of Chemical Dynamics Traced through the Nobel Prizes.* J Chem Ed 79:1297-1303 (2002).
241. Vetter I, Wittinghofer A. *The Guanine Nucleotide-Binding Switch in Three Dimensions.* Science 294:1299-1304 (2001).
242. Wall L, Christiansen T, Schwartz R. *Programming Perl, 2nd Ed.*. O'Reilly (1996).
243. Wand A. *Dynamic activation of protein function: A view emerging from NMR spectroscopy.* Nature Structural Biology 8:926-931 (2001).
244. Wang J, Wolf R, Caldwell J, Kollman P, Case D. *Development and Testing of a General Amber Force Field.* J Comput Chem 25:1157-1174 (2004).
245. Waterman M. *Introduction to Computational Biology: Maps, Sequences and Genomes.* Chapman & Hall, London (1995).
246. Welch M, Chastang J, Yarus M. *An Inhibitor of Ribosomal Peptidyl Transferase Using Transition-State Analogy.* Biochemistry 34:385-390 (1995).
247. Westhof E, Sundaralingam M. *Restrained Refinement of the Monoclinic Form of Yeast Phenylalanine Transfer RNA. Temperature Factors and Dynamics, Coordinated Waters, and Base-Pair Propeller Twist Angles.* Biochemistry 25:4868-4878 (1986).
248. Wider G. *Structure Determination of Biological Macromoleculs in Solution Using NMR Spectroscopy.* Bio Techniques 29:1278-1294 (2000).
249. Wilson D, Szostak J. *In Vitro Selection of Functional Nucleic Acids.* Annu. Rev. Biochem. 68:611-647 (1999).
250. Wimberly B, Brodersen D, Clemons W, Morgan-Warren R, Carter A, Vonrhein C, Hartsch T, Ramakrishnan V. *Structure of the 30S ribosomal subunit.* Nature 407:327-339 (2000).

251. Woese C, Winker S, Gutell R. *Architecture of ribosomal RNA: Constraints on the sequence of "tetra-loops".* PNAS 87:8467-8471 (1990).
252. Woese C. *Translation: In retrospect and prospect.* RNA 7:1055-1067 (2001).
253. Woo M, Neider J, Davis T, Shreiner D. *OpenGL Programming Guide, 3rd Ed..* Addison-Wesley (1999).
254. Woodbury G. *Physical Chemistry.* Brooks/Cole (1997).
255. XFree86.org widgets. *XFree86 Documentation: Specification and Related Documents: Athena Widgets (Xaw).* http://www.xfree86.org/4.4.0/widgets.html (2004).
256. XPLOR website. *X-PLOR HOME.* http://xplor.csb.yale.edu/xplor/ (2005).
257. Youngman E, Brunelle J, Kochaniak A, Green R. *The Active Site of the Ribosome Is Composed of Two Layers of Conserved Nucleotides with Distinct Roles in Peptide Bond Formation and Peptide Release.* Cell 117:589-599 (2004).
258. Yusupov M, Yusupova G, Baucom A, Lieberman K, Earnest T, Cate J, Noller H. *Crystal Structure of the Ribosome at 5.5 A Resolution.* Science 292:883-896 (2001).
259. Yusupova G, Yusupov M, Cate J, Noller H. *The Path of Messenger RNA through the Ribosome.* Cell 106:233-241 (2001).
260. Zagrovic B, Sorin E, Pande V. *Beta-Hairpin Folding Simulations in Atomistic Detail Using an Implicit Solvent Model.* J. Mol. Biol. 313:151-169 (2001).
261. Zavialov A, Ehrenberg M. *Peptidyl-tRNA Regulates the GTPase Activity of Translation Factors.* Cell 114:113-122 (2003).
262. Zhang M, Kavraki L. *A new method for fast and accurate derivation of molecular conformations.* J Chem Inf Comput Sci 42:64-70 (2002).
263. gnuplot website, Williams T, Kelley C, Lang R, Kotz D, Campbell J, Elber G, Woo A. *gnuplot homepage.* http://www.gnuplot.info/ (2005).
264. seqaln website. *U.S.C Sequence Alignment Package.* http://www-hto.usc.edu/software/seqaln/ (2005).
265. van Holde K, Johnson W, Ho P. *Principles of Physical Biochemistry.* Prentice-Hall (1998).
266. wxWidgets. *wxWidgets Home Page.* http:/www.wxwidgets.org (2004).