Molecular Viewer Using Spiegel

Pavani Baddepudi

Masters Project Report

Department of Computer Science

Golisano College of Computing and Information Sciences

Rochester Institute of Technology

May, 2007

Approved By Committee:

_____

Chairman**: Dr.Hans Peter Bischof** Date

_____

Reader: **Dr.Joe Geigel** Date

_____

Observer: D**r.Sidney Marshall** Date

Acknowledgements

I would specially like to thank Dr.Hans Peter Bischof, my project advisor, for suggesting the project topic and providing constant guidance throughout the project period. This project would not have been possible without his patience and support. I would also like to extend my thanks to Dr.Joe Geigel and Dr.Sidney Marshall for agreeing to be the committee members and providing suggestions. Last but not the least I would like to thank my husband, brother and parents whose relentless affection, encouragement and inspiration helped me complete this project.

Abstract

Study of nucleic acids and proteins has evoked great interest for its role in research involving areas such as human genome and drug design. Understanding the core structure and behavior of these protein molecules requires reading and analyzing large amounts of data about atoms derived from a plethora of experiments. In recent years, computer graphics has gained popularity for its ability to translate these large datasets into three dimensional structures providing easy visualization. This project has implemented one such 3-D molecular viewer to display different models of molecules and study their properties. The project was developed in Java3D using the Spiegel visualization framework. The final outcome of this project is software which provides a mechanism to visually analyze protein structure using well known interactive features such as zoom, rotate and translation.

Table of Contents

Table of Figures

1. Introduction

Molecular modeling is the science and art of studying molecular structure and function through model building and computation. [5] Building physical models for larger molecules like proteins has always been difficult due to its complex physical and chemical structures. Use of computer graphics gained popularity amongst scientists in the field of biology for its ability to analyze large datasets and represent them in three dimensional formats to evaluate them. The result of this project is a 3-D molecular viewer developed within the Spiegel framework. Spiegel is a modular and highly extensible visualization framework that was developed by the Computer Science department at RIT to support the display of any three dimensional structures. The work was focused on representing a wide variety of renderings of protein molecules in Spiegel environment to view different aspects of the same model, such as the primary and the secondary structures. Features like rotation, translation, zoom were added to better examine the molecules.

1.1 Structure of Project Report

Section 2.0 discusses the backgrounds of molecular biology and other related topics and offers its connection to this project. It also looks into a few related projects carried out in this field. Section 3 reviews the system design and implementation details in terms of features added and data structures used. Section 4 describes the Java3D Scenegraph used to develop the code. Section 5 specifies the hardware and the software requirements to run the code. Section 6 concludes this report with a summary and conclusion on the work

carried out in this project and lists out the ideas of potential future work in this area.

2. Background Knowledge and Related Work

This project brings together the fields of biology and computer graphics. Since the range of the study of biomolecules is fairly broad, this project limited its study to macromolecules such as nucleic acids and proteins. This project required to have the basic knowledge about the structure of proteins and nucleic acids to be able to model them. The following sections provide little insight on other concepts and topics that were used to build this project.

2.1 PDB files

The data required for building the visualization software was taken from Protein Database (PDB) in this project. PDB is a central resource that contains records obtained from experiments performed by biologists and biochemists all around the world. The datasets in these files were parsed and extracted by the functions in the Spiegel framework.

2.2 CPK Coloring Scheme

Various coloring models can be used to represent the atoms. However CPK model has been considered as the default industry standard. CPK stands for Corey, Pauling, Koltun and has been named after its developers. The table below shows the colors associated with each element. The atoms displayed in this project used the CPK colors.

Carbon	Light Grey
Oxygen	Red
Hydrogen	White
Nitrogen	Light Blue
Phosphorous	Orange
Sulphur	Yellow
Unknown	Deep Pink

Table 1: CPK Colors

2.3 Covalent and Vanderwaals Radius

Technically, the atomic radius, also known as covalent radius, is one-half the distance between nuclei of two of the same atoms that are bonded to each other. [8] This project used this radius to calculate the inter-atomic bonding between adjacent atoms for all the models except the CPK model which uses the Vanderwaal's radii. Vanderwaals radii are used to display relative sizes of the atoms. The radius here is an imaginary hard sphere determined from measurements of atomic spacing between pairs of unbonded atoms in crystals. [10] The table below shows the covalent and vanderwaals radius of each element expressed in Angstroms.

Element	Covalent	Vanderwaals
Carbon	0.68	1.7
Nitrogen	0.72	1.55
Oxygen	0.68	1.52
Sulfur	1.020	1.85
Phosphorous	1.036	1.9
Hydrogen	0.32	1.20

Table 2- Covalent & Vanderwaals radii

2.4 Algorithm for Inter-atomic bonding

Determining the bonds between atoms continues to be a challenge due to different empirical energies acting on them. This is also one of the most important factors required to determine the structure of the molecules. Various research and experiments conducted have resulted in different formulae and controversial conclusions.

The formula used in this project for calculating the atomic bonding is same as the one used in related work such as Chime and Rasmol [9] and is as follows

if (distance (coords[i], coords[j]) between min_max)
Bond exists
else
Bond does not exist

where cords[i] and cords [j] are the coordinates of atom1 and atom2 respectively and min = 0.6f and max = 1.2 f.

2.5 Java3D

The language chosen for development of molecular structure viewer was Java3D for its easy integration with other Java APIs since the Spiegel framework is developed in Java. The language aided the object oriented design that was being implemented in this project and is easy to understand and to implement.

2.6 Spiegel Framework

Spiegel framework has been developed by Computer Science Department to visualize the N body simulations. [6] The framework consists of suite of modules which are easily extensible and reusable. The modules also known as plugins are interconnected together in a pipeline through which the data flows. Spiegel could be executed on both UNIX and Windows platforms with minimum hardware requirements. Because of its flexible and extensible architecture this framework facilitates rapid modeling software development in any field and was therefore chosen for this project.

2.7 Related Work

There are a number of existing applications for molecular visualization, Jmol being the most popular and extensively used.

Jmol: Jmol application and applet is the most popular software amongst the biologists and chemists. It was developed at the Minnesota Supercomputer Center. It uses Java 3D and does not require any additional hardware to view the images. Jmol supports various

files formats as its input – CIF, mmCIF, PDB, Jaguar, etc.[1] . This project has incorporated some of the salient Jmol features into the Spiegel framework.

Rasmol: RasMol is a program for molecular graphics visualization originally developed by Roger Sayle, intended for the visualization of proteins, nucleic acids and small molecules.[7] Rasmol is developed in C language and runs on various platforms.

Other popular software includes Chime, Protein Explorer, JMV, etc.

3. Project Details

There are four primary interconnected plugins in the Spiegel framework that are essential to its operation. **Extractor** plugin performs the function of extracting the required data from the user input for visualization and passes the data to the Visual or Filter plugin. **Filter** plugin is an optional module which is used to prune unwanted extractor data to focus only on the structure of elements the user is interested to see on the display. **Visual** plugin provides the means for analyzing the extracted data (from Filter or Extractor plugin) by producing three dimensional objects from them. **Camera** plugin displays the complete structures on the screen. It also provides controls for the user to move the camera and view the image at different angles and positions in the scene.

The figure below shows how all the components fit into the Spiegel framework and help realize the final image of the molecules on the screen.

[1] Please refer www.jmol.org for more details on the file format supported and what each one represents.

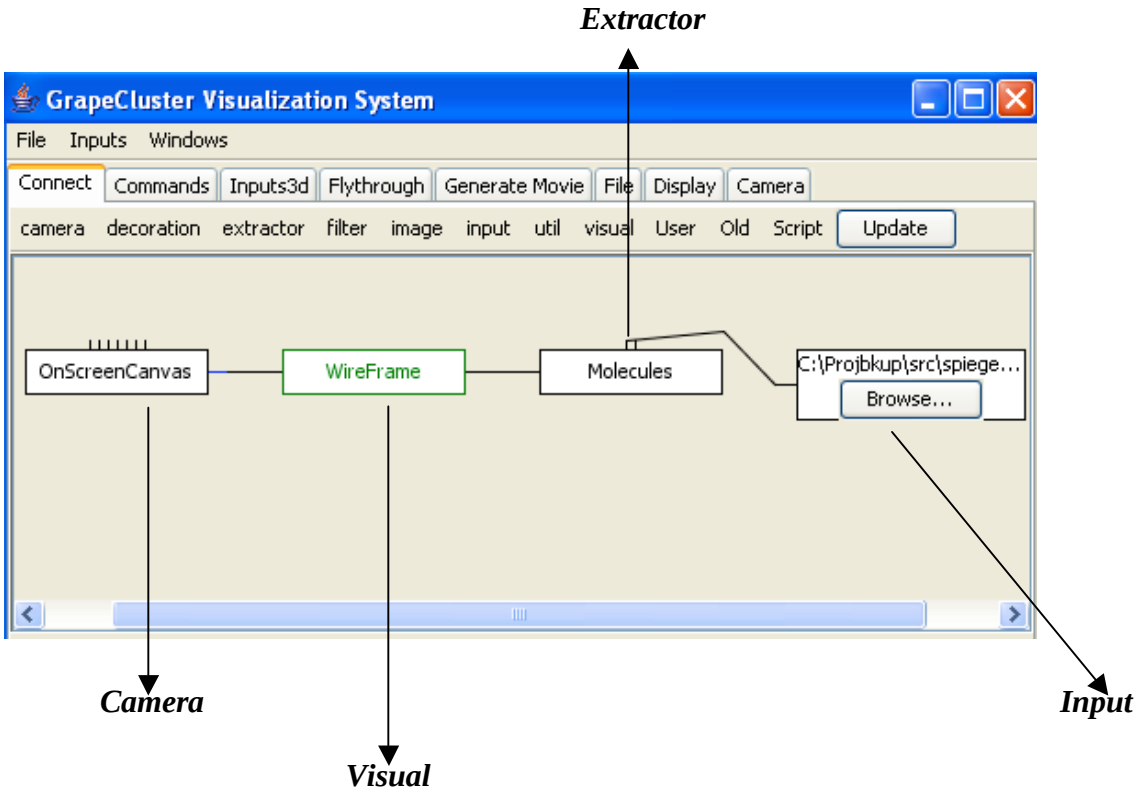

Figure 1: Spiegel Framework

The Spiegel framework provides controls for adding, deleting and connecting the plugins. The Update button reflects the changes to the image in the display whenever any values of the plugins are changed. The entire setup could be saved as a Sprache script which could then be loaded using command line arguments or from the File menu.

The framework provides with functions for each plugin which are overloaded as per the design. While the User Guide in Appendix B discusses the input and the output variables required for each plugin used in this project, the following sections focuses on the implementation details and data structures used.

3.1 Extractor Plugin

The **Molecules** plugin has been designed to accept input files in pdb format only. It parses the pdb files to find all the lines starting with “ATOM”, which contains the X, Y, Z coordinates of the elements. This plugin has various collection classes that store the data relevant for visual representations. HashTable **chainIds** stores the chain information which helps to estimate the backbone structure, the HashMap **atomLis**t contains the atom names and serial numbers while the HashTable **positionList** holds the coordinates of the atoms to be placed in the scene. The class **NeighborList** calculates the distance of an atom between its neighboring atoms and determines their bonding with the atom.

The extractor passes **MoleculeIDMap** as the output to the next plugin which could be a filter or a visual plugin depending on the representation chosen for the display. MoleculeIDMap is a HashMap containing the serial number and name of the atoms.

3.2 Visual Plugin

Visual plugins work on different formats of the molecule to be displayed depending on the user’s selection. The output of all the visual plugins is the **BranchGroup** object which contains the information about the shape, size and color of the atoms. The class **AtomData.java** in the Visual plugin holds information about the color as well as the radii information (both Covalent as well as VanderWaal’s radii) for each element. This plugin contains four different classes that provide four different representations of the molecules.

i) Wireframe Representation

It is the simplest of all models which displays the bonding information between the atoms. This is useful in visioning the structure of the three dimensional image. The vertices of the adjacent atoms are joined by the **LineArray** object. If two atoms A1 and A2 are bonded; half of the distance between the two is represented by the color of element A1 while the other half distance is represented by atom color of A2. All the LineArray objects are added to a single **Shape3D** node. **LineAttribute** object is added to the Appearance element to control the line width and the pattern. The line width of 1.5f is chosen and the pattern is set to PATTERN_SOLID. Shape3D node is finally added to the parent BranchGroup node.

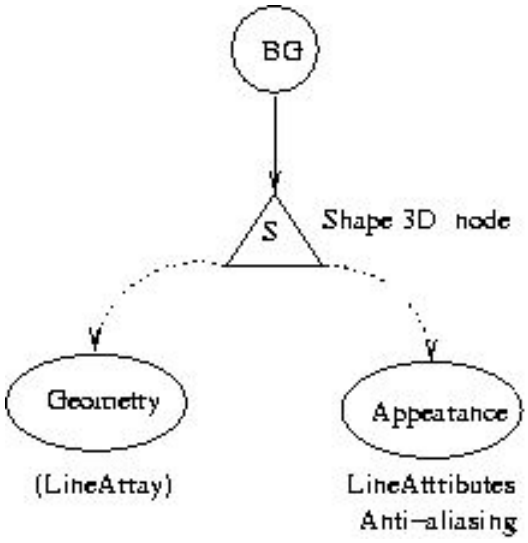

Figure 2 – Content Scenegraph of Wireframe Model

The figures below show the setup in the Spiegel pipeline and the image resulting from this order.

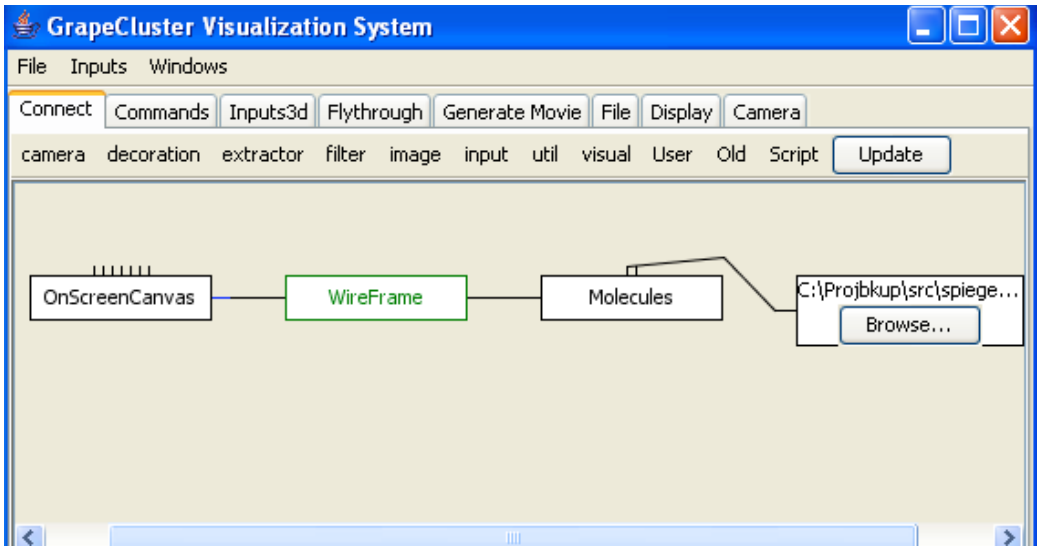

Figure 3 – Spiegel Pipeline for Wireframe Representation

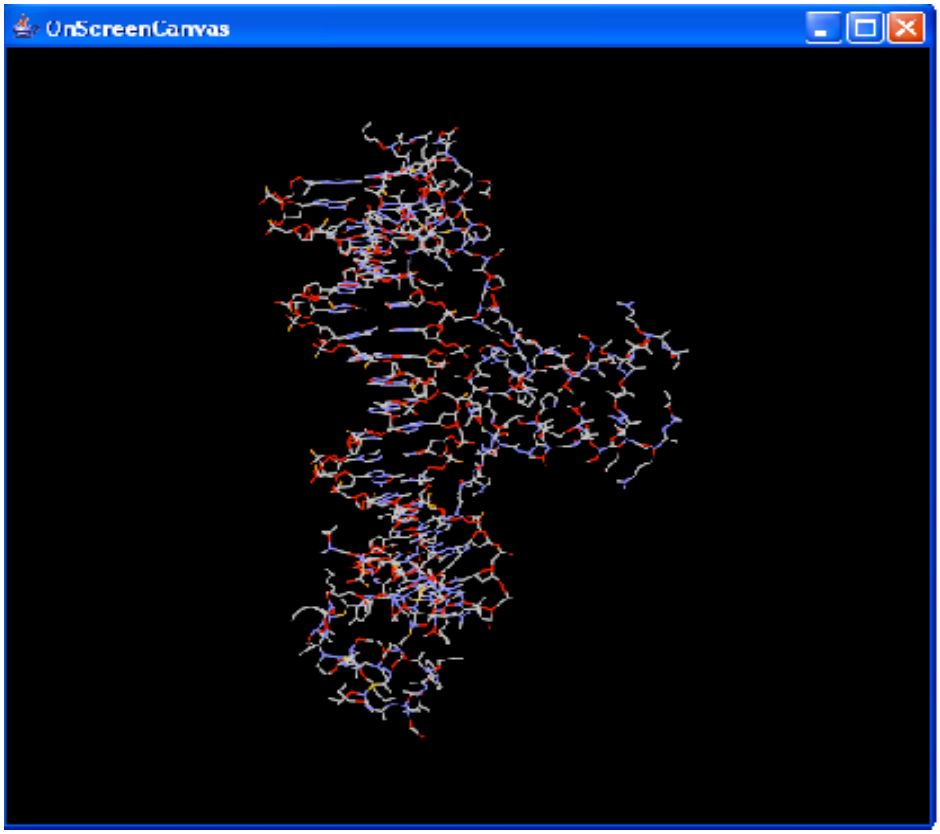

Figure 4– Wireframe Representation of 1D66.pdb

ii) Ball and Stick Representation

This is the traditional ball and stick representation. The bonds are represented in gray color by the **LineArray** object while the atoms are represented in CPK colors by the **PointArray** objects. The appearances of both the objects are controlled by the **LineAttribute** and **PointAttribute** objects respectively.

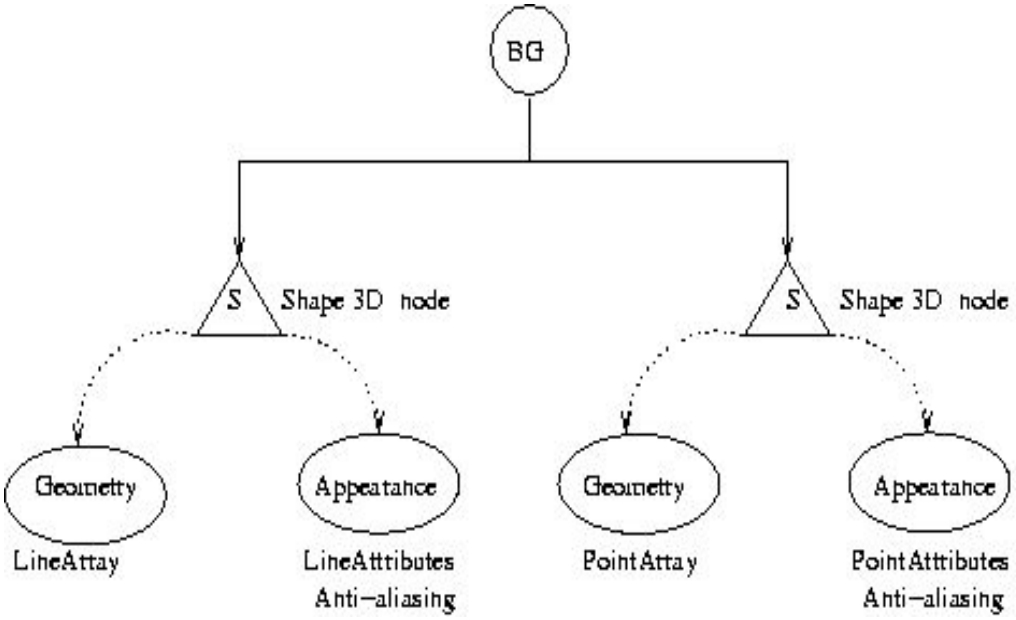

Figure 5– Content Scenegraph of Ball and Stick Model

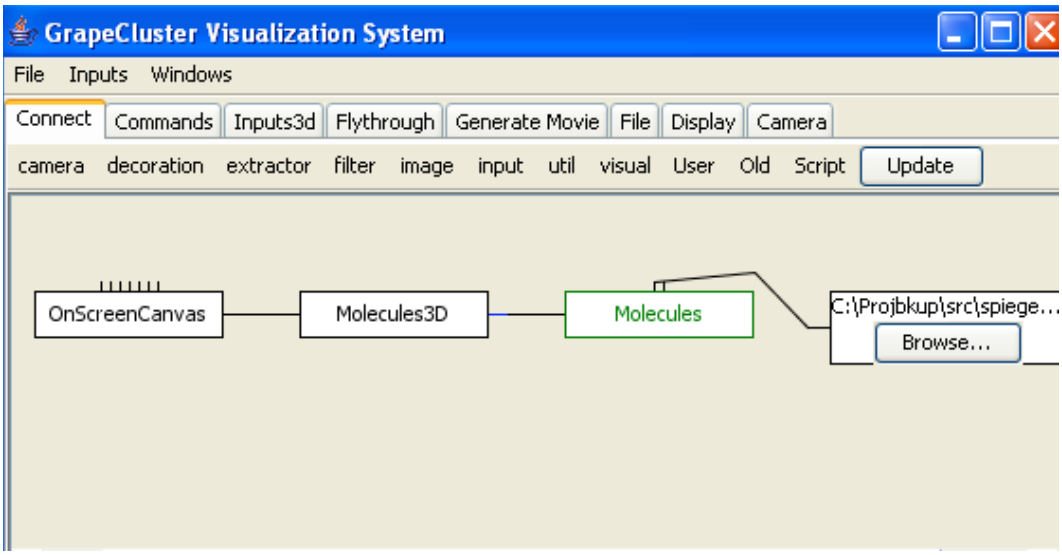

Figure 6– Spiegel Pipeline for traditional Ball and Stick Representation

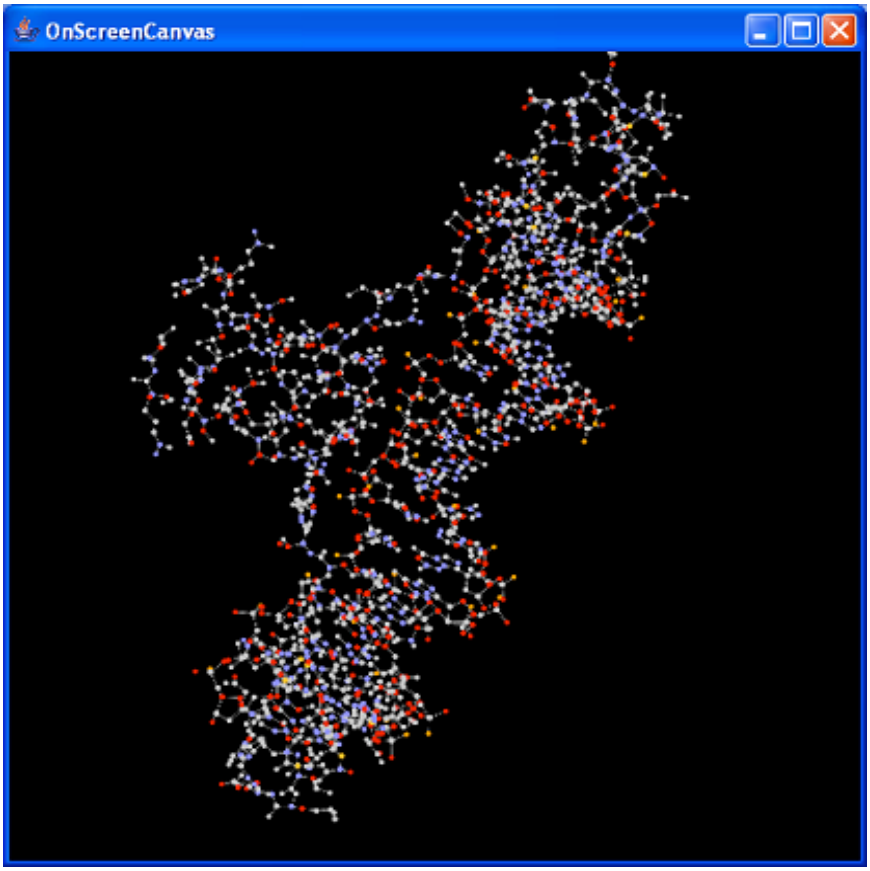

Figure 7: Ball and Stick Representation from 1D66.pdb

iii) Backbone representation

The backbone structure which displays the Secondary structure of the molecule is formed by joining the adjacent phosphate atoms in case of nucleic acids and carbon alpha atoms in case of amino acids. The color chosen for the chains were at discretion and were not bound to any defacto standards. In this model, the coordinates of the atoms are connected by **LineStripArray** object which joins the adjacent atoms as a set of continuous, connected line strips. The width of the lines has been set to 3.0f through the **LineAttribute** object.

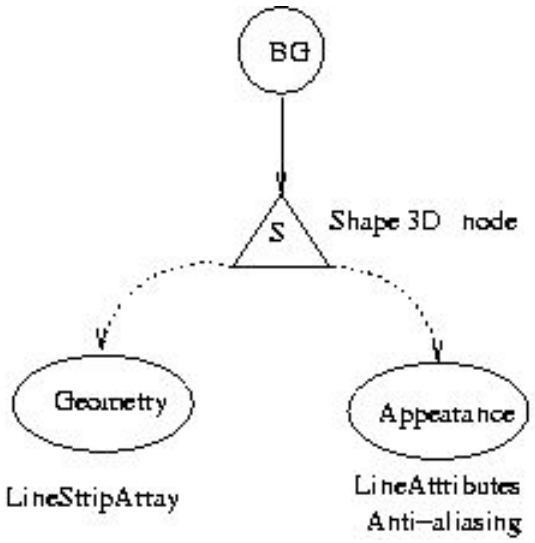

Figure 8– Content Scenegraph of Backbone Model

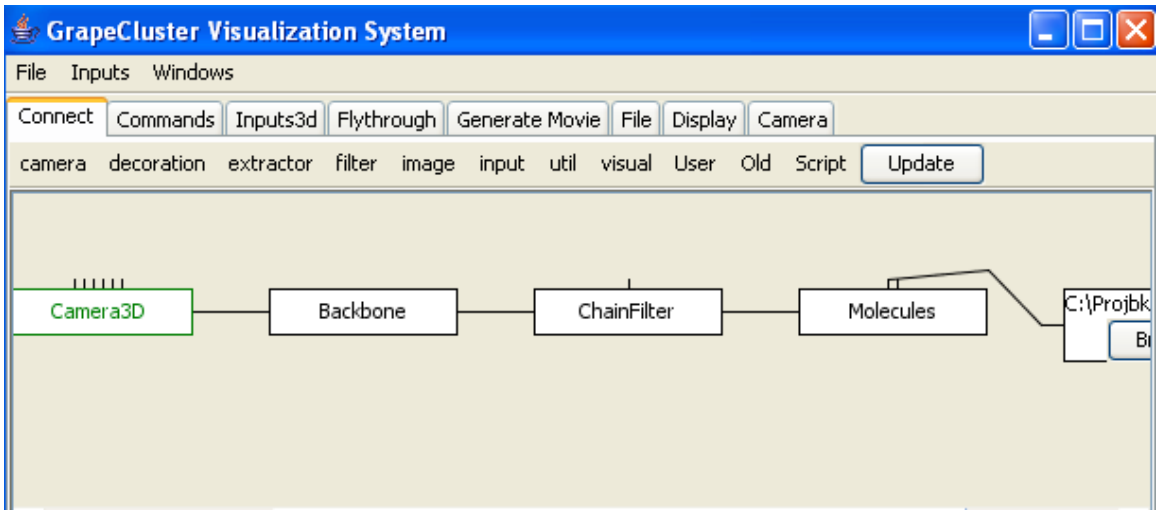

Figure 9– Spiegel Pipeline for Backbone Representation

Figure 10– Backbone representation of 1D66.pdb

iv) Spacefill representation

It is also called the CPK model. Spacefill representation helps to understand the amount of volume the molecule occupies and therefore Vanderwaal's radius has been used for this model. Since the atoms are represented in spheres of different sizes in this model, primitive **Sphere** object of Java3D has been used to represent them. **Material** object has added to the appearance component of the spheres to respond to the lights while **Transparency** has been added to view the intersection between the atoms. The Sphere objects are added to the **TransformGroup** object to translate the atoms to their correct positions

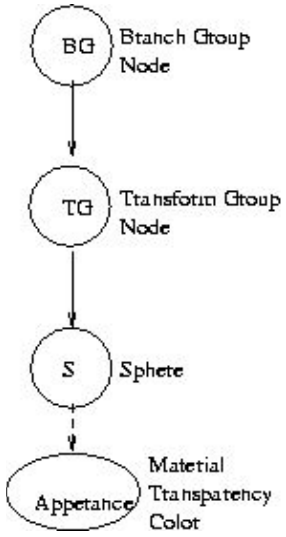

Figure 11– Content Scenegraph of CPK Model

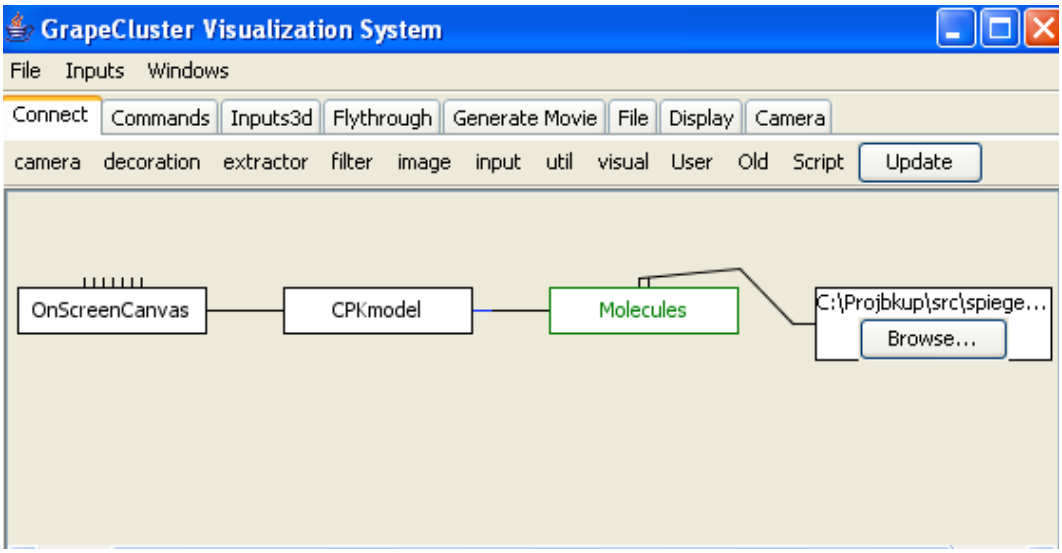

Figure 12– Spiegel Pipeline for CPK Model

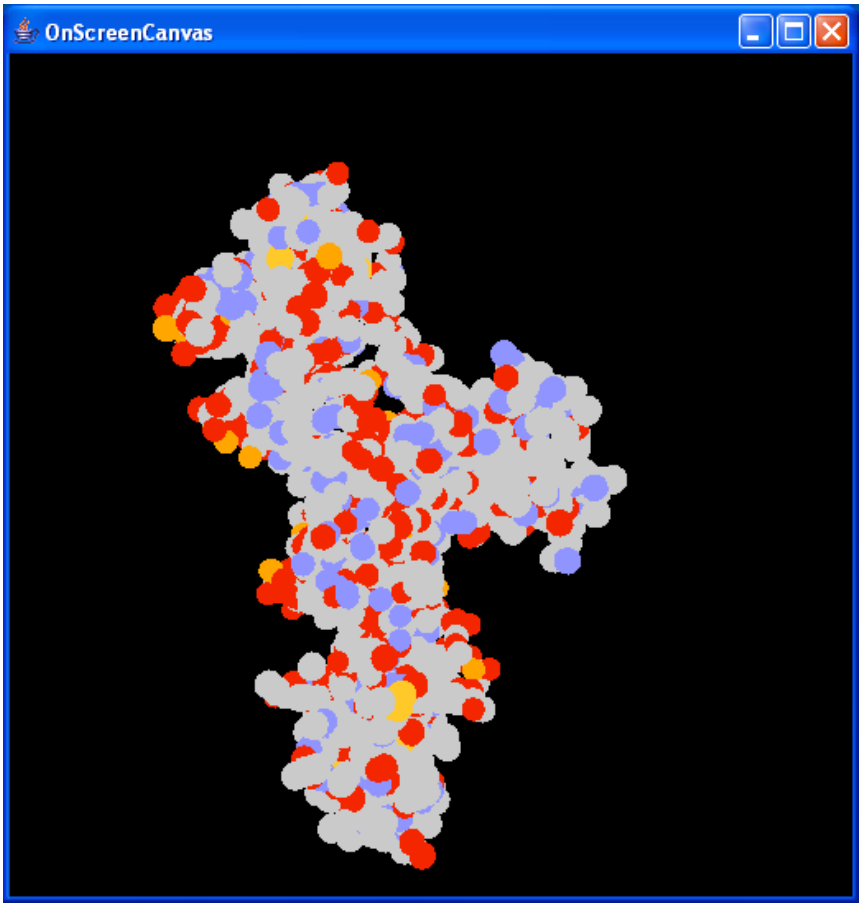

Figure 13–Sphere Representation in Spiegel framework

All the models were scaled to fit the screen. Anti aliasing has been added to attributes of all GeometryArray (Line Array, LineStripArray, and PointArray) objects for smooth picture rendering.

3.3 Camera Plugin

Camera plugin renders the final image obtained from the Visual plugin onto the canvas. Two types of renderings have been developed depending on the type of **Canvas** object used namely on-screen and off-screen canvas. In OffScreen canvas, the actual rendering is done to an off-screen hardware buffer or to a 3D library-specific buffer and only copied to the off-screen buffer of the Canvas when the rendering is complete, at "buffer swap" time. [2] This is useful for batch rendering of non interactive images. Camera3D class implemented uses the Offscreen canvas for the final display. One of the tasks of the project was to make it user interactive. Behavior classes were required for rotating, zooming and translating the images using mouse controls and also for animation. Since the behavior cannot work in conjunction with OffScreen rendering due to scheduling issues, OnscreenCanvas class has been created to incorporate all the interactive functions.

3.4 Filter plugin

In this project, filter (**ChainFilter)** plugin has been applied to view the backbone strands. Here the user has a choice to select to view the DNA strands or the protein strands. The user could also input the chain number to focus on individual strand. The code written under this plugin has been customized for 1D66.pdb file and therefore recognizes only

the chain ids included in this file. The filter plugin outputs chainIdMap which is a SortedMap containing the serial number and the chainId of the atoms.

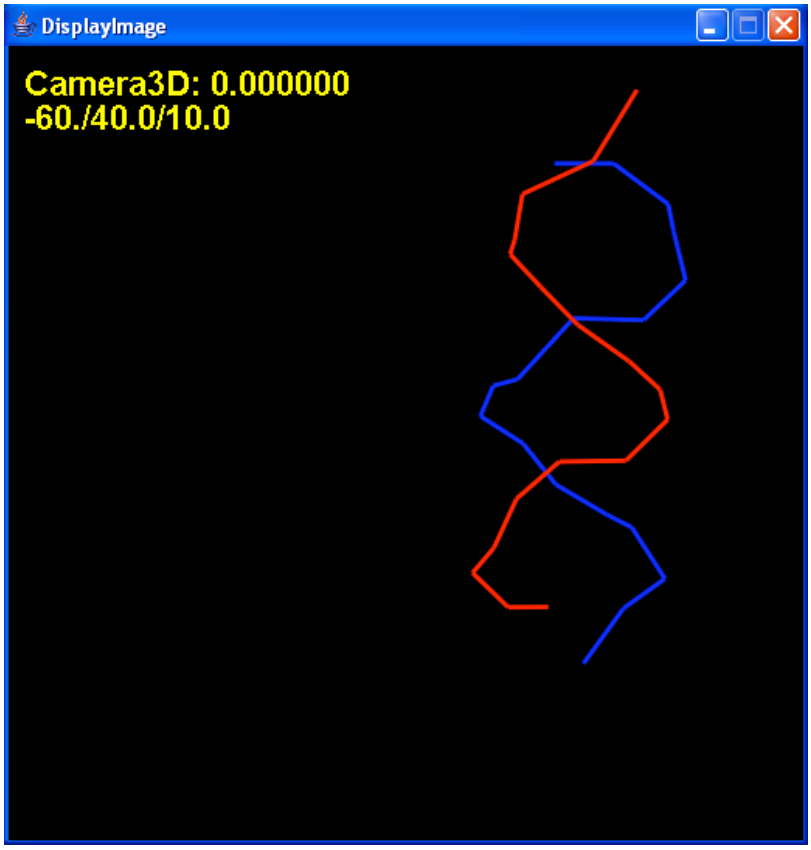

Figure 14– Image displaying the nucleic acids (DNA) strands in 1D66.pdb

4. Scenegraph Structure

All the objects created in Java3D are associated to a scenegraph. A scenegraph can be defined as a tree structure with node and leaf objects arranged in hierarchical manner. Every scenegraph has two superstructures namely VirtualUniverse and Locale. There can be only one Virtual Universe in the project and it defines a distinct area under which all the objects to be viewed are defined. The Locale object provides the reference for objects

in the universe and is also the parent for different branchgraphs. There are two types of branchgraphs – content branchgraph and view branchgraph. Content branchgraph contains nodes that carry information about the structure of the objects to be displayed. View branch concentrates on projection of the objects onto the scene. Fig.6 below shows a typical Java3D scenegraph hierarchy which gives us the basis to discuss the graph structure approach taken in this project.

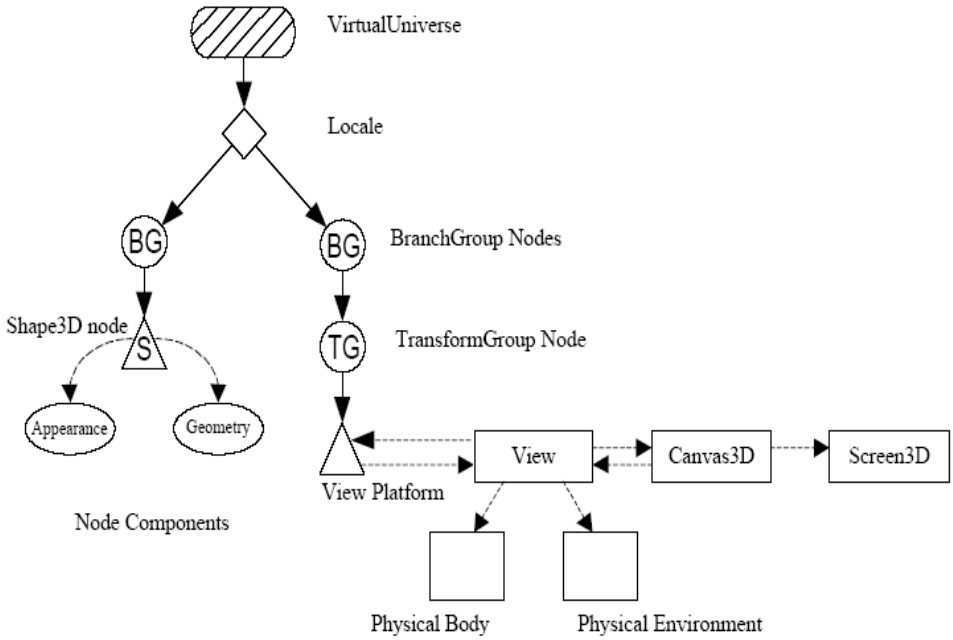

Figure 15- Represents Java scenegraph [4]

The project followed the bottom up approach for execution where the content branch was implemented first followed by the view branch. The Visual plugin provided all the components for content branchgraph while the view branchgraph is formed in the Camera plugin. The visual plugin processed the atom data received from the extractor and assigned shape, size and color to the atoms. Different content branchgraphs were

designed to view different models of the molecule.

The Content branchgraph in this project however has two more nodes compared to the typical model – TransformGroup and BranchGroup. The RotationInterpolator class used to spin the models required that it be attached to a TransformGroup. Therefore TransformGroup node was added above the BranchGroup obtained from the Visual plugin. This TransformGroup node was finally added to another BranchGroup which formed the root of the Content branchgraph. Background and lights for the scene was added to this root BranchGroup. Ambient Lights and Directional Lights were chosen for illuminating the scene.

To be able to view the content branchgraph, view branchgroup was created and added to the Locale in the Camera plugin. The view branchgroup contains two nodes the TransformGroup and the ViewPlatform. The ViewPlatform is referenced by the View object and it contains information about the Canvas to be used for the Display, Screen that contains the Canvas and Physical Environment. [2] The View component contains the viewing transform matrix that is used to define the final view of the image on the screen. Interactive features were added to the view model where the molecules could be manipulated with the predefined mouse input events. This was achieved by adding MouseListener to the Canvas object which allowed a range of moves, rotations and zooming of the structure

The final programming model of this project looks like the one below

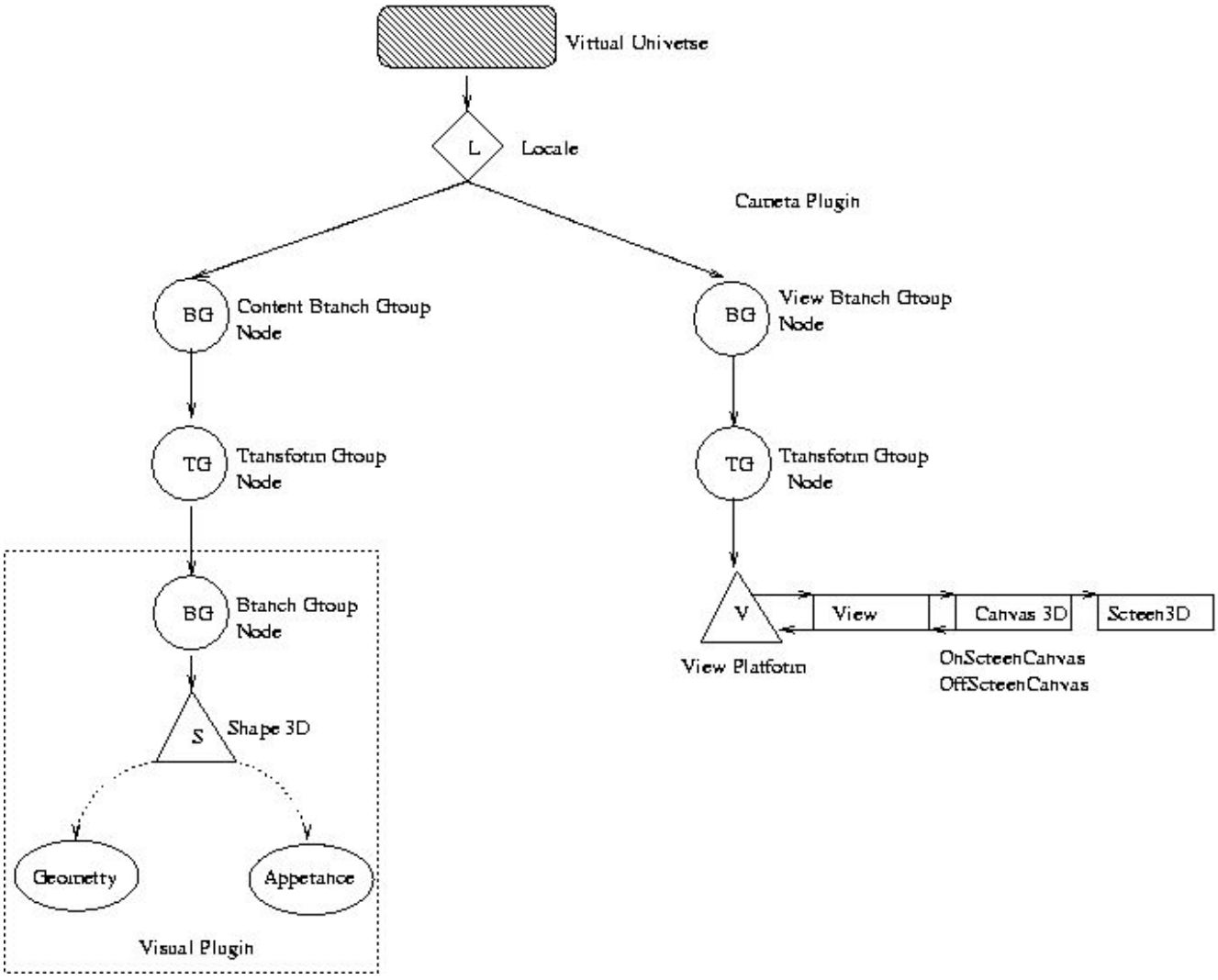

Figure 16– Scenegraph used for this Project

5. Hardware and Software Requirements

Windows XP	Development and Testing was done on Windows platform.
GrapeCluster source code	The source code is freely available on http://www.cs.rit.edu/~grapecluster. This contains the framework on which the project is being developed.
JDK 1.5, Java 3D 1.3.1	Java Development kit , Graph based 3D Java API for developing
MS Office	Microsoft Office tools for documentation of the project
Cygwin	Collection of tools that to emulate Unix client on Windows box. Available freely on www.cygwin.com. Cygwin was used to run the source code. However this is optional and the code can be run from Windows environment also.

6. Deliverables

1. Spiegel.jar - This source file contains the code for Spiegel framework. All the functions implemented and mentioned throughout the thesis have been placed under the functions folder.
2. Data – The pdb files used for rendering the 3D images.
3. Images – The final images of models implemented in the project
4. Project Report – Word Document containing the details about the design and

development of the project

5. PowerPoint presentation – to be presented during defense.

7. Conclusion

The objective of this project was to design and construct a 3D molecular modeling system with user interactive features. The initial stage of this project was spent in understanding the architecture of Spiegel framework and the correlation between different modules .The design stages involved studying the structure of proteins and understanding the raw data in pdb files to extract all the atom information. Based on this knowledge the molecular viewer was a designed. The 3D rendering required Java3D skills which were acquired during different phases of the project.

Spiegel framework helped achieve this tool much faster than programming in traditional environment. The design helped me to concentrate only on the 3D rendering part. Handling events like adding, deleting or updating the modules or integration of the Java3D with Swing to develop the application was already provided in the framework.

This work required areas of expertise in many areas unfamiliar to me and posed challenges at every stage which was overcome by consulting the professor and people online in Java forums. This project succeeded in its main objective of implementing different display models and functions to manipulate them. This work has laid foundation for the simulation projects that can be carried out using these models.

8. Future Work

This project lays a solid foundation on which future work can be carried out for adding more features, improving the user interface or adding dynamic simulations. One of the areas for enhancement is to extend the support for multiple file types like CIF, MDL, etc. Giving user a choice to change the colors of the atoms and select atoms based on name would have helped to focus on atoms of particular interest in the complicated structure and study their behavior.

Two main things that remained unsatisfactory in this project implementation are

i) Animation feature: Though the rotation interpolator was implemented to achieve spinning in models, the modular structure of the Spiegel framework made it difficult to pass the information about the center of the molecule. Therefore the axis of rotation which passes through the center of the molecule had to be hardcoded which worked fine for few models only.

ii) Bond Calculation: This has been an area of ongoing research with various formulas developed to calculate the inter-molecule and intra-molecule bonding. Exploring and employing other theories to build the models and comparing them with the existing ones could have helped in building a more accurate model.

9. Bibliography

1) www.cs.rit.edu/~grapecluster

2) Java 3D API tutorials at www.sun.java.com

3) www.rcsb.org

4) Daniel Selman: “Java 3D programming”

5) Tamar Schlick: Molecular Modeling and Simulation: An Interdisciplinary Guide

6) “Hans-Peter Bischof, Jonathan Coles: A Movie Is worth More Than a Million Data Points"

7) http://www.umass.edu/microbio/rasmol/rasbonds.htm

8) http://en.wikipedia.org/wiki/Covalent_bond

9) http://www.umass.edu/microbio/rasmol/rasbonds.htm

10) http://en.wikipedia.org/wiki/Van_der_Waals_radius

A. Sprache Scripts

Spiegel framework provides inbuilt scripting called Sprache. Below is the example of the Sprache script that implements the Wireframe model.

```
function new visual.WireFrame WireFrame
function new extractor.Molecules Molecules
function Molecules set file null
function Molecules set time 0.0
function new camera.camera3d.OnScreenCanvas OnScreenCanvas
function OnScreenCanvas set location -8.0 5.0 5.0
function OnScreenCanvas set lookat 5.0 0.0 0.0
function OnScreenCanvas set up -1.0 0.0 1.0
function OnScreenCanvas set size 512 512
function OnScreenCanvas set time 0.0
function OnScreenCanvas set rotate false
function OnScreenCanvas set spin false
function new input.File File
function File set file C:\Projbkup\src\1D66.pdb
function connect WireFrame output object to OnScreenCanvas input objects
function connect Molecules output Molecules to WireFrame input atoms
function connect File output file to Molecules input file
thisPointInTime 0.0
updateview
```

B. User Guide

grapecluster.jar could be downloaded from www.cs.rit.edu/~grapecluster under the Software section. The files are extracted to a directory "C:/Project/src/Spiegel". The **root** variable in Makefile.opts under the directory needs to be changed pointing to "C:/Project/src/Spiegel" folder. All the java files under this directory can be complied using the command **"make all"** in Cygwin environment. "**make runD**" will finally run the compiled program.

The sections below detail the input options available under each plugin in the framework which help achieve the final image on the screen.

1) Extractor Plugin

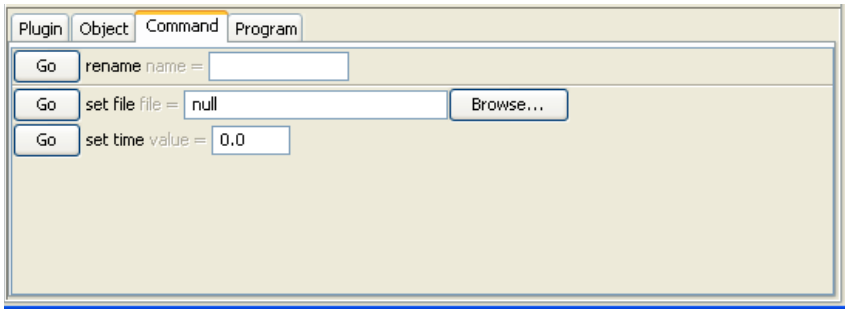

Figure 17– Molecule3D options

Clicking on Molecules3D button under the Extractor in the main menu adds Molecules3D to the main program. All the changes made to the plugin variables however take effect only when the GO button next to the settings and Update button is pressed.

Window's "delete" button can be used to delete any plugin on the main window.

The Molecules3D plugin accepts pdb files as input which could be loaded by clicking on the browse button under the **set file** tab. The "time" variable would be useful for future simulation projects and has been set to 0 for this project.

2) Filter Plugin

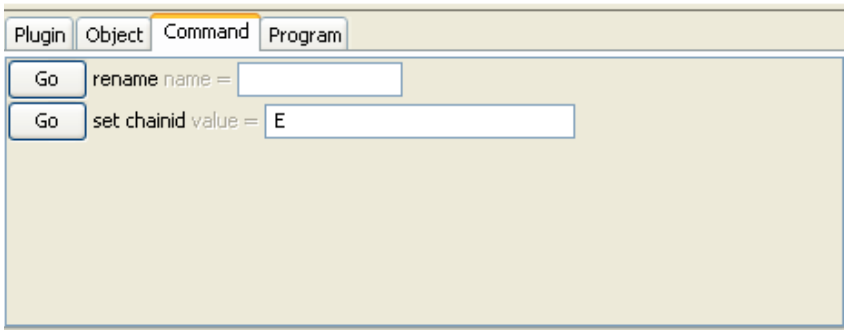

Figure 18– ChainFilter Options

The ChainFilter plugin has been customized for the 1D66.pdb file and **set chainid** takes in the following characters as input.

1) **P** – Amino Acid chains

2) **N** – Nucleic Acid chains

3) **A, B** – Individual DNA strands

4) **D, E**- Individual Protein strand

3) Visual Plugin

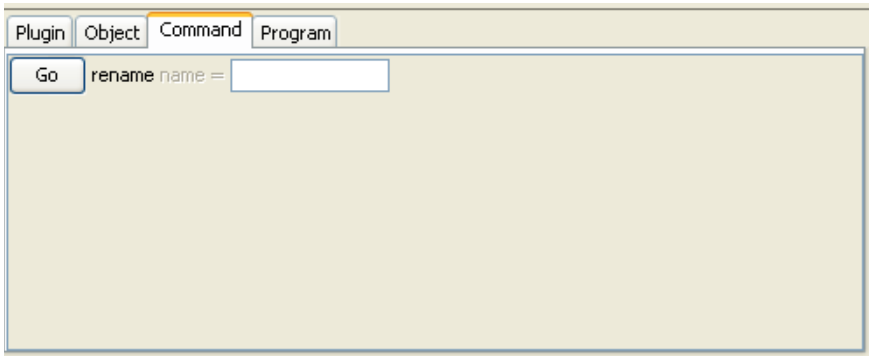

Figure 19– Visual Plugin Options

Four different types of Visual Plugins could be added to the pipeline by selecting them from the Menu.

4) Camera Plugin

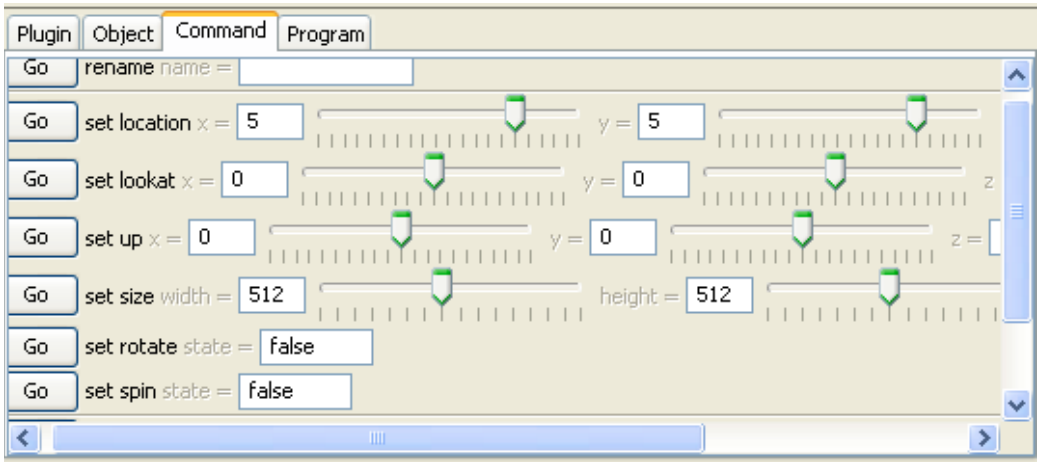

Figure 20– OnscreenCanvas/ Camera3D Plugin options

The options **set location**, **lookat** and **up** together determine how and where the image is

displayed on the screen. The canvas is set to a default size of 512x512. The dimensions however could be changed by changing the **height** and **width** options provided under the **set size** command.

The set rotate option rotates the image by 90 degrees in Y direction. This was added for to rotate images in camera (Camer3D) using Off-Screen Canvas. The images displayed using the Onscreen Canvas can be rotated, translated and zoomed in any direction using the mouse controls. The **set spin** option when set to true invokes the animation feature implemented in this project. The image completes the 360 degrees rotation in 4 seconds. The spinning can be disabled by changing the option to false.