TR-0856

A Three-Dimensional Animation System for Protein Folding Simulation

by
M. Akahoshi, K. Onizuka, M. Ishikawa,
& K. Asai (ETL)

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191~5

Institute for New Generation Computer Technology

A Three-Dimensional Animation System for Protein Folding Simulation

MASAYUKI AKAHOSHI KENTARO ONIZUKA
MASATO ISHIKAWA
Institute for New Generation Computer Technology(ICOT)
1-4-28, Mita, Minato-ku, Tokyo, 108 Japan
akahoshi@icot.or.jp, onizuka@icot.or.jp, ishikawa@icot.or.jp
FAX +81-3-3456-1618

KIYOSHI ASAI
Electrotechnical Laboratory(ETL)
1-1-4, Umezono, Tsukuba Ibaraki, 305 Japan
asai@etl.go.jp
FAX +81-298-58-5939

Abstract

We have developed a new computer language that describes three-dimensional graphical objects in visual simulation. The language, 3D-Talk, is a powerful tool for visualizing the process of protein folding simulation. 3D-Talk is designed as an object-oriented language, with which users can create, manipulate, and edit graphical objects using commands with a syntax similar to that of a natural language.

We developed two novel molecular biology applications using 3D-Talk. The first, **ProView**, *is a protein visualization tool.* **ProView** *converts PDB(Protein Data Bank) data into 3D-Talk statements. The second is a visual simulation system for protein folding. It simulates the motion of protein folding using simulated annealing techniques and Hidden Markov Models.*

In short, 3D-Talk serves as a powerful animation tool.

1 Introduction

Visualization systems are playing an ever-more important role in the field of molecular biology and drug design [4]. Such systems help biologists and engineers study the three-dimensional structures of biochemical molecules. However, researchers and engineers in these fields generally spend more time building the visualization modules than they do building data analysis modules, because the programming of a visualization module is so complex that biologists, who are not trained in programming, find it overly demanding. A user-friendly visualization scheme is, thus, strongly desired. Such a system would be able to, say, display a sphere when a user merely types "Draw a sphere."

The PostScript[1] language contributed greatly to visual communications. This is a simple interpretive programming language with powerful capabilities and considerable flexibility. However, since this language is optimized to enable machines to readily generate the code, the statements of this language are quite difficult for users to understand, offsetting its desirable features.

As an easy-to-use visualization environment, we developed a description language, 3D-Talk, and its interpreter, **3DView**. 3D-Talk was designed as an object-oriented language with a simple grammar.

For example, the statement

```
Ball is a sphere;
Ball radius is 5;
Ball draw;
```

represents a sphere object with a radius of 5. (Fig.1)

We built an interpreter, **3DView**, which creates, displays, and manipulates graphical objects according to their description in 3D-Talk. Since our main purpose in designing this interpreter was to develop a vi-

[1]copyright©1985,1986,1987,1988,1990 Adobe Systems Incorporated.

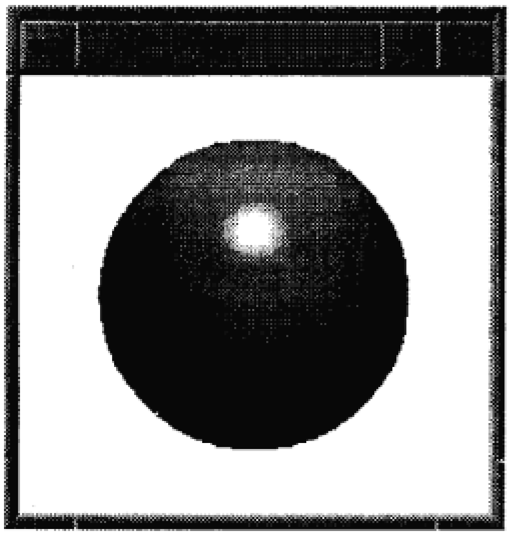


Figure 1: Sphere Object

sual folding simulation system, this interpreter would have to be suitable for animation. In this paper, we will show how this language and interpreter are suitable for the animation of folding simulation. Then, we will briefly illustrate a protein folding simulation system that uses our graphical system.

The organization of this paper is as follows. In Section 2, we discuss the prerequisites for visualization environments and describe the motivation for our approach in 3D-Talk. In Section 3, the specifications of 3D-Talk are described in detail. In Section 4, the features of the interpreter, **3DView**, are explained. In Section 5, we briefly introduce two applications. The first **ProView**, is a protein visualization system, and the second is a protein folding simulation system. We show how effectively the features of 3D-Talk work for them. In Section 6, the conclusion, we also describe our future works.

2 Motivation and Policies

We have been studying how a protein folds into a three-dimensional structure. We needed a graphical system that can visualize a protein structure to see this folding process. Although there are several commercial tools for molecular visualization, none are suitable for our purposes, because of the following reasons.

1. It is impossible for users to add new functions to such tools, because these tools are provided as binary compiled code, with no source code.
2. The descriptions of graphical objects are internally processed and the results stored as binary data. This makes these tools less flexible.
3. The protein structure database "PDB (Protein Data Bank)" is revised frequently, where the description format is changing with each revision.

We overcame these problems by developing a new visualization scheme. Ideally, a visualization environment should display the objects that a user wishes to visualize. Also, the system should be able to accept new PDB formats or descriptions after only minor modification by the user. Here, we consider the following criteria regarding the realization of such an environment:

1. Which kind of system is better, an interpreter or a compiler ?
2. How should graphical objects be represented ?
3. What kind of grammar is preferable for the description language ?
4. How flexible is the system for network communication ?
5. What kind of extended features are needed ?

The first criterion concerns the flexibility of the visualization environment. Graphical objects should be represented and manipulated by specific description. Graphical libraries, such as GL and PHIGS [1] can display and manipulate objects quickly because the description is compiled into machine-readable binary code. But, the compiled code is almost impossible to modify and requires the use of many control parameters.

On the other hand, interpreters such as PostScript have considerable flexibility. Any Postscript description is written, transmitted, and interpreted as ASCII characters. Therefore the description can always be printed and read. This feature is quite convenient for users to read, edit, and write descriptions in this language. This is also useful for storing descriptions into files, and for transferring files between different kinds of machines using different operating systems. This enhances machine independence. Thus, we decided to build a system which *interprets* the description because we consider flexibility to be of the highest importance.

The second criterion concerns the representation of graphical objects, (e.g., sphere, pillar, and box [5]).

Each graphical object should have sufficient parameters to determine its properties, such as shape, color, size, position and orientation. Each graphical object should be able to change its state, such as visible, invisible, wired, solid, or not solid, according to commands input by the user.

In this case, exploiting the concept of object-orientation allows us to deal with these properties systematically. The set of parameters representing a graphical object is separated into two parts. One is the common part while the other is the private part, as shown below:

object type	common	private
line	color, magnify,	width, style
pipe	move, location, draw, hide	radius

Graphical objects are hierarchically classified, in our system, into several classes, according to the properties of each object. The common part is inherited from the class above. The private part defines its own properties. In this case, the size of the statement is greatly reduced by the concept of inheritance in 3D-Talk.

The third criterion concerns the grammar of a description language. It is difficult for users to learn a language if they are unfamiliar with its grammar. Since PostScript programs are normally generated by applications rather than the user, PostScript's grammar is not necessarily easy for users to master. Another reason why PostScript grammar is presents difficulties to the user is that its grammar is optimized for the interpreter to parse. When large amounts of graphical data are handled, this approach achieves high performance.

However, we believe a simple grammar to be better than a complicated one, because the purpose of our system is to provide an easy visualization environment where users can easily read, edit and write description language statements. We designed an English-like language , or a kind of Pidgin English. The Pidgin English-like sentences are immediately understandable, ever to inexperienced users.

The fourth criterion concerns the transmission of data. The performance of an application using network communication depends largely on the data transfer speed of the network. Since the data transfer speed is determined by the hardware and protocol of the network system, the application cannot control the speed. Thus, to build a high-performance application, we have to minimize the amount of data being handled by the network.

The amount of data required to animate graphical objects is quite large, because the visualization system needs graphical data for each frame of the motion. 3D-Talk solves the problem of large amounts of data. **3DView** has an internal state that stores the states of all created graphical objects. This means that the 3D-Talk interpreter does not have to receive all the graphical data, instead only having to receive the differences in state from the previous frame to display the next state.

The fifth criterion concerns the number of complex features. Users usually prefer an environment with many complex features, because they believe that this would make it easy to visualize what they wish to do. But, in reality, the statements required to realize such an environment are too complicated to learn. We decided that the features of a user-friendly environment would be compact and easy to understand. In designing an application, users should make up more complex features only as needed.

3 Specifications of 3D-Talk

Objects in 3D-Talk are classified into two types. These are graphical objects and non-graphical objects. Graphical objects are those that can be displayed upon the input of 'draw'. Non-graphical objects are those that define the configurations and environment of the graphical objects.

This language supports the following classes of objects:

graphical	dot, line, sphere, cylinder, box, character, pillar, pipe, pyramid, polygon, cone, rectangle, group
non-graphical	characters, character strings, numbers, locations, lists, materials light

Figure 2: All class name

Message 'a' or 'an' is sent to the so-called background object with the argument specifying the class of object to be created. The background object creates the object according to the message. The created object should be referenced by an identifier. The message 'is' binds the identifiers and the created objects, as below:

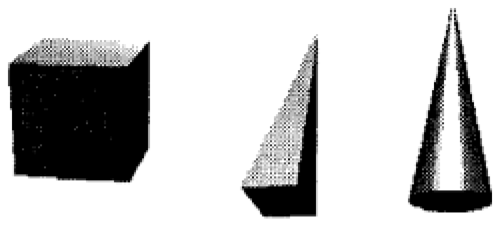


Figure 3: Graphical Object Samples

```
Ball is a sphere;
Pole is a pipe0;
Case is a box;
```

where 'Ball' is an identifier indicating the created object by the message 'a sphere'.

We often need to group several objects to manipulate them together as a single complex object. The object class 'group' is introduced for this purpose. Fig. 4 shows a statement representing a group object. The sixty-line statement defines a group object of Glycine residue which consists of four atoms and three bonds. (Hydrogen atoms are not included, since PDB ignores them.)

A group object usually consists of several subgroup objects. The statement, shown on the left in Fig. 5, defines the hierarchical relationship indicated on the right.

The syntax of this language is roughly represented by the BN form below.

```
<statement>  ::= <expression> ";"
               | <message> ";"
<expression> ::= <expression> <message>
               | <object> <message>
<object>   ::= <object> member_selector
             | <article> class_name
             | variable
             | literal
<article> ::= "a"
            | "an"
<message> ::= message_selector
            | message_selector <argument>
<argument> ::= "(" expression ")"
             |  <coordinate>
             |  <list>
             |  variable
             |  literal
<coordinate>::= "{"number "," number "," number"}"
<list>    ::= "[" <argument> [ "," <argument>] "]"
```

member selector	radius depth top bottom height division string font size width style points color
message selector	is draw drawall hide hideall wire wireall transform moves location rotate rotation from to alpha append bind cdr car magnify antialiases into up push pop smooths nurbs shines colors speculer insert element value delete quit sync

"variable" should be a character string that is not reserved by the system. Fig.2 describes "class_name". "number" is a numerical value.

A 'message' here can be thought of as corresponding to a verb in a natural language, and it sometimes requires an argument, corresponding to the object of the verb. Some messages to certain classes of objects return a value, after which the returned value can act as an argument for another message.

Each object may have several slots for subobjects. A slot is referenced by its name and is used as an internal variable of the object. If the object is a sphere, each of its properties such as the radius and color is stored in a slot of the sphere object. The slot name is a message name. This means that the subobject is referenced by the object name and slot name, as below.

```
TheBall is a sphere;
TheBall radius is 5.0;
```

where 'radius' is the slot name of the internal variable whose value represents the radius of the sphere object.

4 3DView: Graphical Description Interpreter

The **3DView** visualization module is an interpreter of 3D-Talk that displays graphical objects according to their description in 3D-Talk. Users can visualize objects they want to simulate by writing statements in 3D-Talk and sending those statements to the **3DView** module. Users can move, rotate, and scale the displayed objects by using the mouse. The control panel provides optional operations. Frequent visualization environment configurations can be defined in the initialization file '.3dview'.

Fig.6 shows the main mouse operations supported by this system. The displayed objects are rotated, resized, and moved using these simple mouse operations.

- To move the object, place the cursor on the window, hold down the left mouse button, then drag the object (Fig.6(A)).

```
Root is a group;
push Root;
Residue is a group;
push Residue;
N is a sphere;
N color is :Nitrogen;
N radius is :NitrogenRadius;
N moves{0.433000,0.750000,0.500000};
N2 is a pipe0;
N2 top is :BondRadius;
N2 bottom is :BondRadius;
N2 color is :Nitrogen;
N2 from{0.433000,0.750000,0.500000};
N2 to{-0.106500,1.016000,0.077500};
CA is a sphere;
CA color is :Carbon;
CA radius is :CarbonRadius;
CA moves{-0.646000,1.282000,-0.345000};
CA1 is a pipe0;
CA1 top is :BondRadius;
CA1 bottom is :BondRadius;
CA1 color is :Carbon;
CA1 from{-0.646000,1.282000,-0.345000};
CA1 to{-0.106500,1.016000,0.077500};
CA2 is a pipe0;
CA2 top is :BondRadius;
CA2 bottom is :BondRadius;
CA2 color is :Carbon;
CA2 from{-0.646000,1.282000,-0.345000};
CA2 to{-0.977000,0.708500,-0.727500};
C is a sphere;
C color is :Carbon;
C radius is :CarbonRadius;
C moves{-1.308000,0.135000,-1.110000};
C1 is a pipe0;
C1 top is :BondRadius;
C1 bottom is :BondRadius;
C1 color is :Carbon;
C1 from{-1.308000,0.135000,-1.110000};
C1 to{-0.977000,0.708500,-0.727500};
C2 is a pipe0;
C2 top is :BondRadius;
C2 bottom is :BondRadius;
C2 color is :Carbon;
C2 from{-1.308000,0.135000,-1.110000};
C2 to{-1.921000,0.086000,-1.121500};
O is a sphere;
O color is :Oxygen;
O radius is :OxygenRadius;
O moves{-2.534000,0.037000,-1.133000};
O1 is a pipe0;
O1 top is :BondRadius;
O1 bottom is :BondRadius;
O1 color is :Oxygen;
O1 from{-2.534000,0.037000,-1.133000};
O1 to{-1.921000,0.086000,-1.121500};
pop;
pop;
Root moves{0.646000,-1.282000,0.345000};
Root drawall;
```

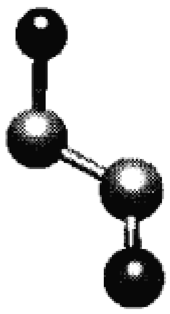


Figure 4: Group Object: Glycine

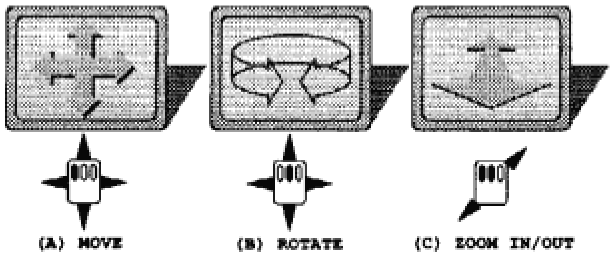


Figure 6: Mouse Operation

- To rotate the object, hold down the middle mouse button, then drag the object. It continues to rotate when you release the mouse button (Fig.6(B)).
- To zoom in on the object, hold down the left and middle mouse buttons, then drag the object down (Fig.6(C)).
- To zoom out from the object, hold down the left and middle mouse buttons, then drag the object up (Fig.6(C)).
- To open the control panel, hold down the right button, pull down the menu, and select "Panel Open/Close".

The ideas for these mouse operations are borrowed from the sample programs of *SGI GL*. These operations act on all displayed objects together. It is impossible to manipulate each individual objects. The control panel shown in Fig.7 is, thus, provided to manipulate individual objects. We can select an object to be manipulated and then send a message to that object, either by typing the message into the message window or by using the mouse to manipulate the iconized elevators and handles.

This system was developed on a SGI (*Silicon Graphics, Inc.*) IRIS-4D system using Graphics Library (*GL*) [2].

3D-Talk and **3DView** have been released as ICOT Free Software entitled 'Protein-View'. For de-

[2]Silicon Graphics, IRIS, 4D and the Graphics Library are trademarks of Silicon Graphics, Inc.

```
Group is a group;
Group Subgroup0 is a group;
Group Subgroup1 is a group;
Group Subgroup2 is a group;
Group Subgroup0 Ball0 is a sphere;
Group Subgroup0 Ball1 is a sphere;
Group Subgroup0 Ball2 is a sphere;
```

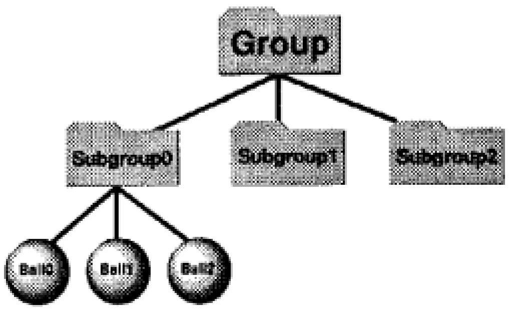
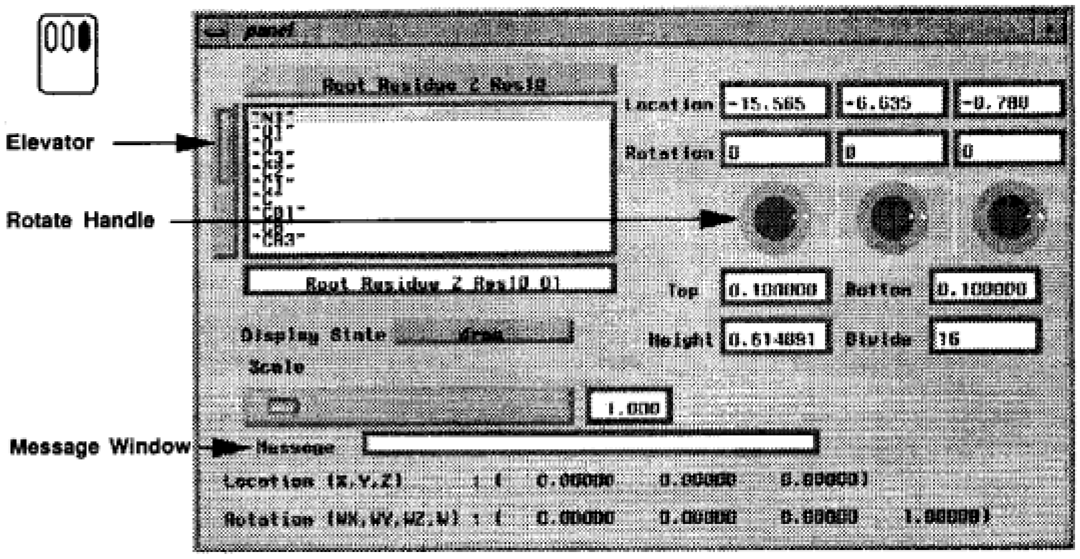


Figure 5: Hierarchical Relation of Group Objects



```
Root Residue 2 Res10
"N1"
"O1"
"O"
"C3"
"C3"
"C1"
"N"
"C"
"CB1"
"CB"
"CA3"
Root Residue 2 Res10 O1
Location  -15.565   -6.635   -0.780
Rotation  0         0        0
Top     0.100000   Botton  0.100000
Height  0.614891   Divide  16
Display State  Atom
Scale  1.000
Message
Location (X,Y,Z)        : (  0.00000   0.00000   0.00000)
Rotation (WX,WY,WZ,W)   : (  0.00000   0.00000   0.00000   1.00000)
```

Figure 7: Mouse Operation(control panel)

tails, please access anonymous FTP, connecting to ftp.icot.or.jp.

5 Applications

We have developed several novel molecular biology applications using 3D-Talk. Two are introduced below: a protein visualization tool **ProView** and a folding simulation system.

5.1 ProView

PDB (*Protein Data Bank*) contains three-dimensional conformation data for about 1250 proteins. One of the most troublesome problems in reading PDB files is that they contain large quantities data in the wrong format, and frequently introduce new ways of representing new kinds of data. This

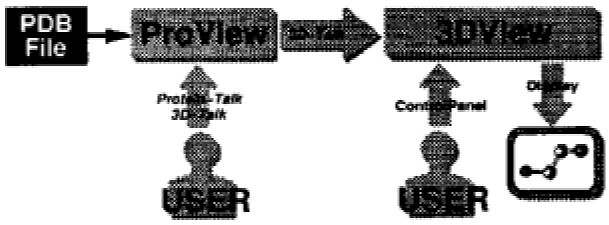


Figure 8: A visualization environment using ProView

is chiefly because the PDB format is not particularly systematic. Therefore, the system should pay careful attention to reading PDB files.

ProView visualizes the conformation of proteins in PDB by generating 3D-Talk. The features of this application are as follows.

1. **ProView** shows secondary structures [3], α-helices, β-sheets, and γ-turns, by using appropri-

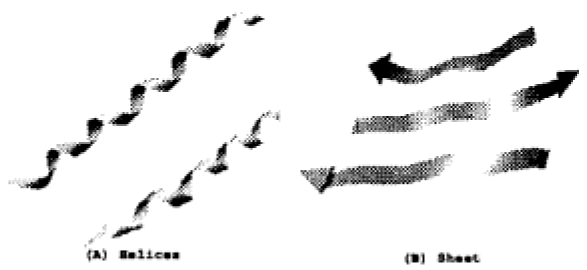


Figure 9: Second Structures Displayed by **ProView**

ate graphical objects. An α-helix is represented by a spiral belt, a β-strand is represented by a belt where a triangle indicates the direction of the strand, as in Fig.9. The site of each secondary structure is determined according to the secondary structure table in a PDB file.

2. **ProView** displays the residue number and residue symbol of each amino acid residue.

3. **ProView** changes the mode of representation. Possible modes are solid mode, wire mode, ribbon mode, and C^α mode (Fig.10). In solid mode, atoms are represented by solid spheres, with chemical bonds being represented by solid cylinders. In wire mode, atoms and chemical bonds are represented by wires. The ribbon mode represents the topology of a global conformation as a belt. Fig. 12 shows a Hemoglobin with four chains. Each of the chains is represented by a different display mode in **ProView**.

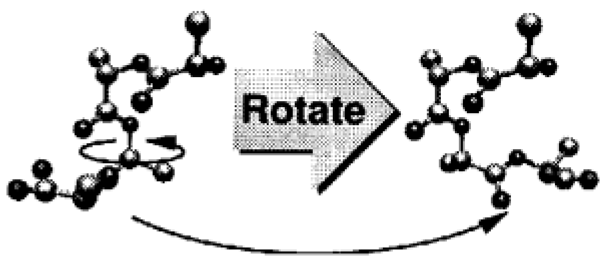


Figure 11: Changing the Dihedral Angle

4. We can change the dihedral angle of a chemical bond to change the conformation of a molecule. To do this, we send the message 'rotate' with an argument denoting the dihedral angle to the object denoting the chemical bond. Fig.11 illustrates the process of rotating the dihedral angle of the chemical bond between Alanine's N atom and C^α. This operation is done by the statement below.

```
a bond from N to CA rotate 120
```

These functions are realized by using group objects. We can use these functions by using Protein-Talk, an operation language that is a specialized version of 3D-Talk.

5.2 Folding Simulation System

There are several difficulties in developing a visualization system for protein folding [7]. The first is how to display the three-dimensional structure of a protein. The second is how to represent the folding process.

We overcame these difficulties by exploiting the special capabilities of **ProView** for protein representation and the innovative functions realized in Protein-Talk. With **ProView** and Protein-Talk, the only statements required for changing the display of protein conformation are bond rotation angles.

Here, we should briefly explain our folding simulation method. We use the simulated annealing [6] technique for the folding simulation. The protein conformation of each stage is translated into MLD (i.e., multi-level description [8]). Then, the fitting score of the conformation is calculated by HMMs (i.e.,Hidden Markov Models [2]) as the probability of the primary sequence of the protein having the MLD.

MLD represents a protein conformation with several symbolic sequences of multiple levels of abstraction. Each symbol in the sequence denotes the class of abstructed topology of subconformation with the size specific to the level. Low-level sequences of this description represent fine structures with high resolution, while high-level sequences represent abstruct structures with low resolution.

We train the HMMs to be the predictors of MLD symbols from the primary sequences. The probabilities of the primary sequences having conformations for certain MLD symbols at each level can be calculated from the HMMs. This means that for a given protein conformation, we can calculate the fitting score from the primary sequence of the protein.

Our folding simulation system generates a set of Protein-Talk/3D-Talk code from MLD symbols in each iteration and the folding process is displayed by **3DView** interpreting the 3D-Talk code.

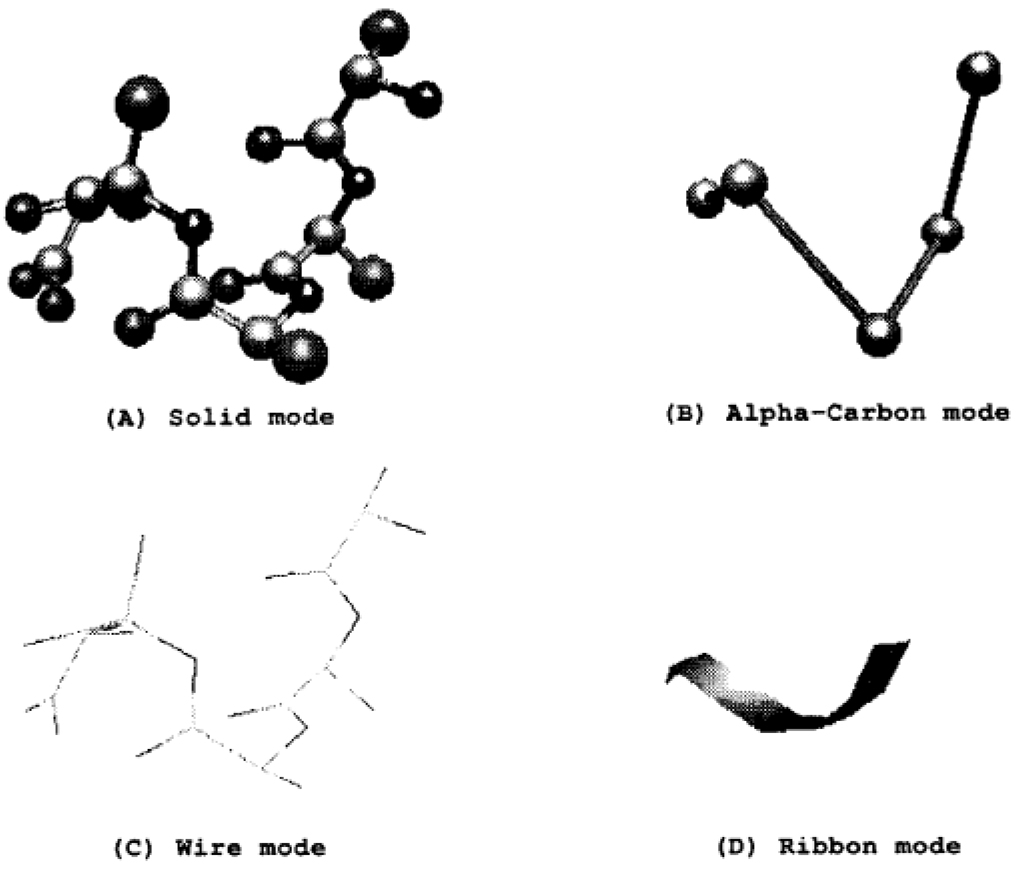


Figure 10: **ProView** Display Modes

6 Conclusion

We developed a new computer language, 3D-Talk, and its interpreter, **3DView**, as an easy-to-use visual simulation environment. Since the language is designed as an object-oriented language and its statements look like English sentences (they appear as a kind of Pidgin English), users, including inexperienced programmers and biologists, can easily create, manipulate, and edit graphical objects.

We have developed several 3D-Talk applications. **ProView** is a special visualization tool for protein conformation. Our protein folding system utilizes **ProView** and 3D-Talk to visualize the process of folding simulation. 3D-Talk enables us to develop innovative graphical application systems.

In the future, we intend to make the interpretation process faster. The speed of interpretation is critical in this kind of graphical system. Several problems in parsing 3D-Talk codes should be overcome to enable the system to read 3D-Talk code faster. Several new functions should be added to our system to reduce the amount of code transferred between applications. At the same time, we should make the statements used in 3D-Talk look more like natural languages to enable biologists to write code without difficulty.

We hope that our novel graphical language and its interpreter provide an agreeable environment for molecular visualization in the field of biology and drug design.

Acknowledgment

The authors would like to thank Toshio Tange at NEC Scientific Information System Development Co. Ltd. for designing and developing parts of 3D-Talk, as well as Akihiko Konagaya at NEC Co. Ltd, for providing fruitful discussion and advice.

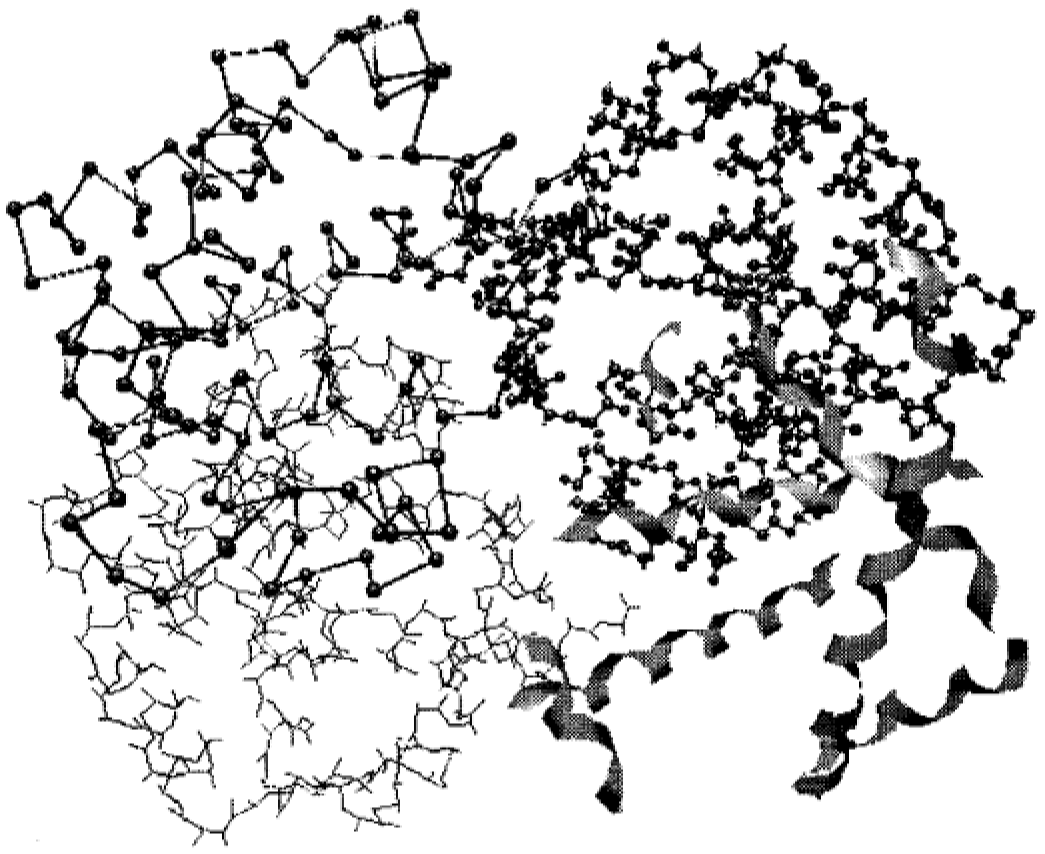


Figure 12: Protein Structure: 4HHB

References

[1] S.S. Abi-Ezzi, A.J. Bunshaft 1986. *IEEE Computer Graphics & Applications, 6. 12 1986.*

[2] K. Asai, S. Hayamizu and K. Onizuka 1993. "HMM with Protein Structure Grammer". *Proc. of 26th Annual Hawaii Int'l Conf. on System Sciences vol. 1*: 783-791.

[3] C. Branden and J. Tooze 1991 *Introduction to Protein Structure*. New York: Garland Publishing, Inc.

[4] Feldmann, R.J. The design of computing systems for molecular modeling.*Ann. Rev. Biophys. Bioeng.*, 5, 477-510, 1976.

[5] J.D. Foley, A.van Dam, S. Feiner and J.D. Hughes 1990. "Representing Curves and Surfaces". *Computer Graphics Principles and Practice, 2nd ed., 1990*

[6] M. Ishikawa, T. Toya, M. Hoshida, K. Nitta, A. Ogiwara and M. Kanehisa 1993. "Multiple Sequence Alignment by Parallel Simulated Annealing". *Computer Applications in the Biosciences Vol.9 No.3.*

[7] Lila M., Gierasch and Jonathan King 1990 *Protein Folding: Deciphering the Second Half of the Genetic Code.* American Association for the Advancement of Science 1990.

[8] K. Onizuka, K. Asai, M. Ishikawa and S.T.C. Wong 1993. "A Multi-Level Description Scheme of Protein Conformation". *Proc. of 1st Int'l Conf. on Intelligent Systems for Molecular Biology.*

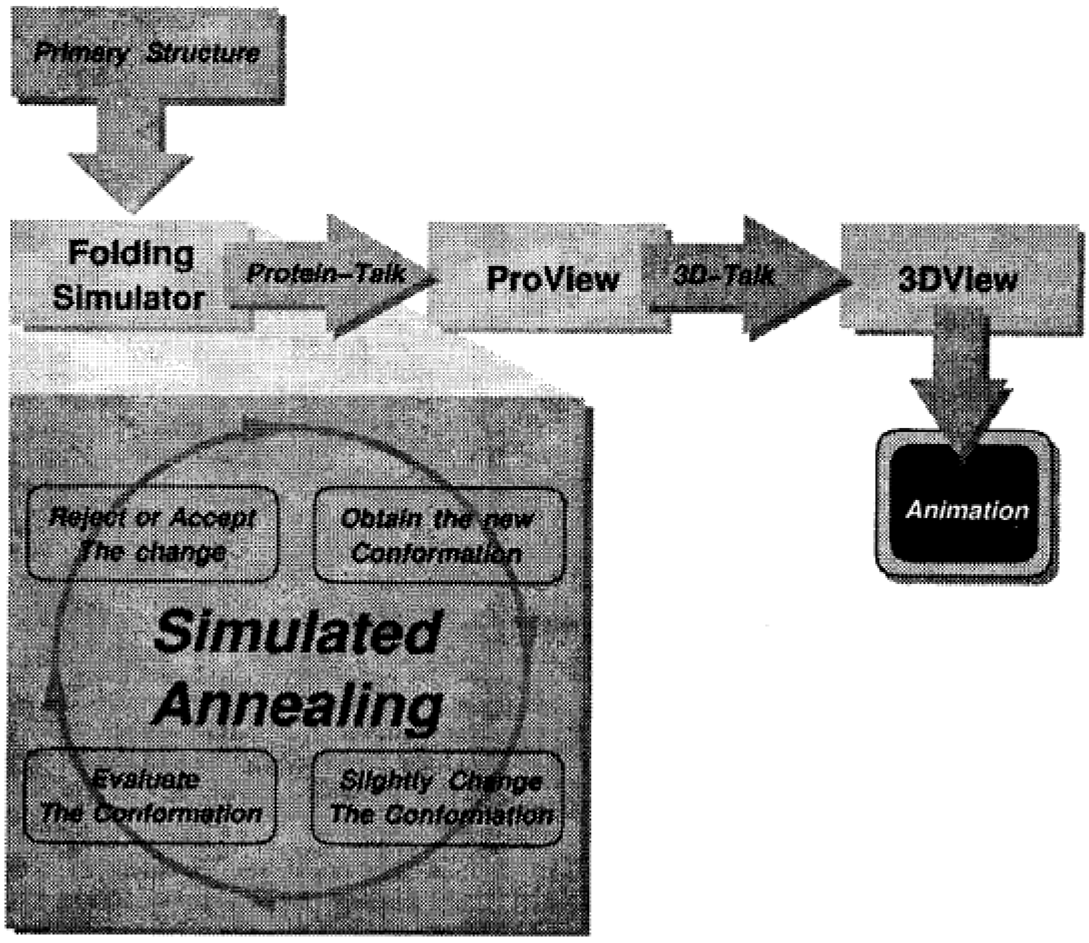

Primary Structure
Folding Simulator
Protein–Talk
ProView
3D–Talk
3DView
Animation
Reject or Accept The change
Obtain the new Conformation
Simulated Annealing
Evaluate The Conformation
Slightly Change The Conformation

Figure 13: Folding Simulation System