Journal of Computer-Aided Molecular Design **18:** 401–419, 2004.

BODIL: a molecular modeling environment for structure-function analysis and drug design

Jukka V. Lehtonen[a], Dan-Johan Still[a], Ville-V. Rantanen[a,b], Jan Ekholm[a], Dag Björklund[a], Zuhair Iftikhar[a], Mikko Huhtala[a], Susanna Repo[a], Antti Jussila[b], Jussi Jaakkola[b], Olli Pentikäinen[a], Tommi Nyrönen[c], Tiina Salminen[a], Mats Gyllenberg[b,d] and Mark S. Johnson[a,*]

[a]*Department of Biochemistry and Pharmacy, Åbo Akademi University, Tykistökatu 6A, FIN-20520 Turku, Finland;* [b]*Department of Mathematics, University of Turku, Matematiikan laitos, FIN-20014 Turun yliopisto, Finland;* [c]*CSC, the Finnish IT Center for Science, P.O. Box 405, FIN-02101 Espoo, Finland;* [d]*Current address: Rolf Nevanlinna Institute, Department of Mathematics and Statistics, FIN-00014 University of Helsinki, Finland*

Received 20 April 2004; accepted in revised form 10 September 2004

Key words: density docking, molecular visualization, sequence comparisons, structure comparison, structure modeling

Summary

BODIL is a molecular modeling environment geared to help the user to quickly identify key features of proteins critical to molecular recognition, especially (1) in drug discovery applications, and (2) to understand the structural basis for function. The program incorporates state-of-the-art graphics, sequence and structural alignment methods, among other capabilities needed in modern structure–function–drug target research. BODIL has a flexible design that allows on-the-fly incorporation of new modules, has intelligent memory management, and fast multi-view graphics. A beta version of BODIL and an accompanying tutorial are available at `http://www.abo.fi/fak/mnf/bkf/research/johnson/bodil.html`

Introduction

As the biological sciences enter what has been referred to as the 'post-genomic' era, where the focus has shifted to the comparison of genomes and detailed investigation of the encoded proteins, the comparison and analysis of sequences and three-dimensional structures have become routine aspects in molecular biology. Very often, direct visualization of molecular structures and their relationship to the linear sequence and to ligands they bind is necessary to interpret and understand the detailed biological functions of proteins, often revealed only indirectly by wet-lab experimentation.

Five years ago we began to develop a graphical interface for our own programs and to use this graphical interface to ease the development and use of novel software. The result today is the Bodil Molecular Modeling Environment, which provides flexible and convenient integration of protein comparison and modeling tools coupled with high-quality molecular graphics. In Bodil we sought to do basic tasks (e.g., alignments, display, high-similarity structure modeling) well, while fully realizing that it was impossible to accomplish all of the tasks that are available in commercial programs. We aimed to make a straightforward user interface for some of the tasks that are difficult to achieve in commercial and other academic packages – most often tasks related to data manipulation or access to frequently used procedures.

*To whom correspondence should be addressed. Fax: +358-2-215-3280; E-mail: johnson4@abo.fi

Furthermore, we wanted Bodil to be written in such a way that it would be easy to introduce new modules and to modify and enhance the program as needs change. Our ultimate goal was to produce a quality set of tools useful for protein structure–function analysis and applicable to ligand design, and whose features could evolve with future desires.

Bodil consists of a core program and a set of modules that perform different tasks. These tasks include reading/writing sequence and structure files, making multiple sequence alignments, alignments of three-dimensional structures, graphical display of structures, estimating the coordinates of a protein structure, 'protein modeling', and so on. The core of Bodil provides for common data storage and management of plug-in programs. The plug-ins present data in different ways, e.g., alignments, structure, surface features, relationships, etc. The change of the common data in one plug-in notifies the other plug-ins via the core program. Thus, one can highlight sequence identities, differences, amino acid properties (e.g., hydrophobicity, polarity, size, etc.), motifs in the sequence alignment, and their location on the three-dimensional structure is immediately shown. Likewise, interesting structural features are linked back to the residues in the sequence alignment.

The graphical structure view editor makes it easy to mock-up a complicated view of proteins and any bound ligand molecules, where, for example, different parts of the structures can be displayed simultaneously as opaque and transparent surfaces, the secondary structure as ribbons, and any portion of the structure as ball-and-sticks, CPK or as wire frameworks. Bodil can read in density grids specifying the ideal location for binding chemical groups [1], chemical probe grid maps from AutoDock [2], GRID [3] and electron density from X-ray crystallography or cryo-electron microscopy (cryo-EM). We have used a plug-in devised by us within Bodil to dock X-ray coordinates into low-resolution electron density obtained from cryo-EM. Grid density data can be displayed as contours, ranges or as iso-energy values, colored appropriately.

The current generation of mid-priced graphics cards for desktop machines can display and rotate surfaces without the annoying delay seen previously even on older, high-priced graphics workstations. At the present time, a fast personal computer running under Linux is an ideal solution, but the program is nearly fully functional under the Microsoft Windows operating system, too. Bodil can use hardware stereo mode, for which graphics cards and X-windows support is available for the Linux operating system.

Methods

Bodil – design strategy

The initial design goal of the program Bodil was to create a software package that would visualize molecules with high quality graphics even on Intel-based PC computers, which would present the user with a simple and intuitive, yet powerful, graphical user interface, and that should be easy to expand by adding new functionality. The main design task was to choose an appropriate data structure for the biochemical data and to develop an efficient method for incorporating useful algorithms into the program. The result is a modular design, where the main executable contains only the common data and the algorithms are encapsulated in modules that are physically separate, dynamically loaded libraries. This produces independent modules, which only require access to the common data. New functionality can be added to the program simply by adding modules; the existing program does not need to be changed. Similarly, changing a module requires only the recompilation of that single library. There is also a run-time performance benefit from this modular design: modules are loaded into memory only if they are used and unloaded after use to release the memory.

In Figure 1 we show the program components and the modules currently implemented. The modules have been grouped by their purpose: visualization tools, computational algorithms, parsers, and utilitarian procedures. The visualization modules employ different techniques to show the data graphically and allow interactive modification of the data. The parser modules convert data between the program's internal representation and external file formats. The set of algorithms includes computational procedures both to assist visualization and to analyze proteins and molecular interactions. The utility modules

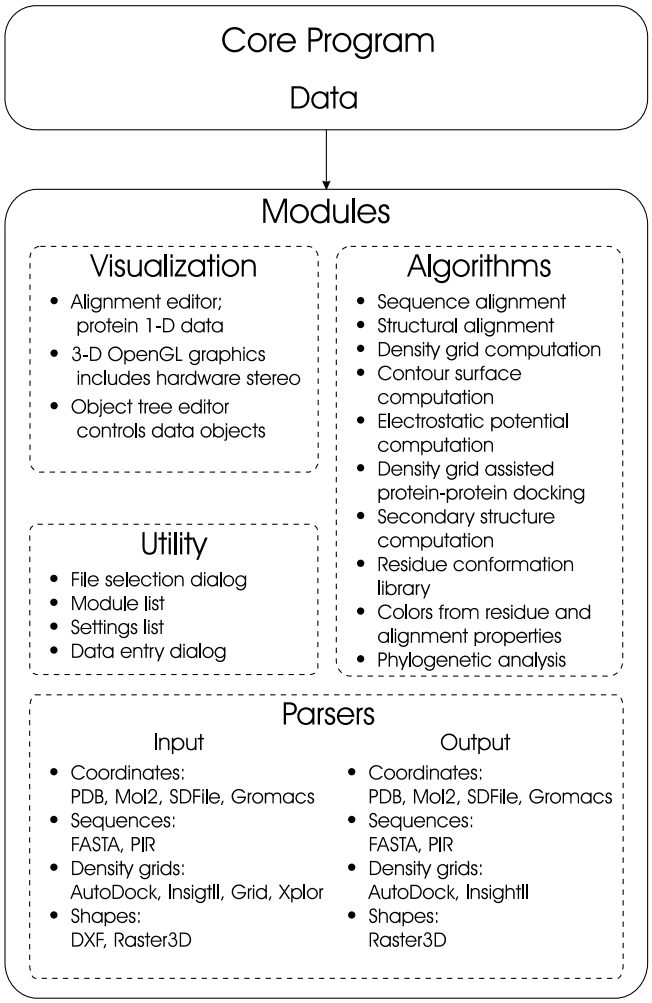

Figure 1. Program components. The program is divided into the core binary, which holds the data and module libraries that contain the computational algorithms and the user interface. The modules are grouped by their task. Some of the algorithms are implemented within the visualization modules, and some computational modules do have their own user interface for selecting operating parameters. The density grids are three-dimensional arrays of discrete spatial data, which can be, for example, electron densities, electrostatic potentials, or a force field. Besides molecules and density grids, the program also handles alignments and arbitrary geometric objects.

provide helper functions, such as the selection of the files for data import.

The data structure

Biochemical data contain both clearly defined physical entities and the relationships between those entities. Some entities, like molecules, are clearly composed of smaller parts – the atoms. These higher-level abstractions of molecules are usable alone; a protein sequence describes the one-dimensional properties of a protein and only implies the existence of an atomic structure. Thus, it is a sufficient representation of the protein for sequence comparison purposes. A chemical bond is an example of a stronger dependency, since it must explicitly refer to defined atoms. It is rather straightforward to use an object oriented data structure to represent such data in the form of an object tree. A composite design pattern [4] describes a way to construct an object tree, and each molecular entity can be modeled by either of the two basic types – leaf and composite – defined by the pattern. The leaves are always terminal nodes in the tree and the composites can be internal nodes. An internal node represents a

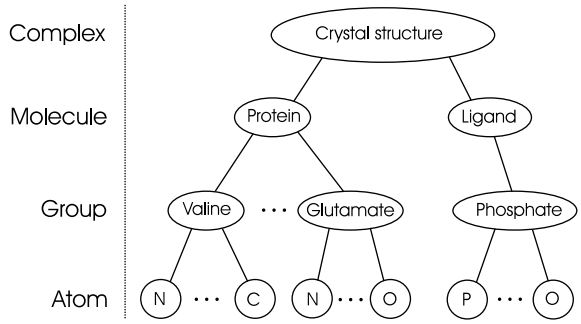

Figure 2. Hierarchy of molecular data. On the left we list major levels of the hierarchy; on the right is an example tree organized according to the hierarchy.

larger structure that is composed from smaller objects (leaves). Thus, an operation performed on a composite node does not change that node directly. Instead, the operation is automatically performed on every node directly under the composite node in the tree (the child nodes). Since a child node can be a composite node, the resulting recursion asserts that the operation is performed for each leaf node within the subtree, rooted at the composite node on which the operation was initiated. Thus, the state of the composite node is changed indirectly, as the state is a composition of the states of the leaf nodes. In addition, composite biochemical entities, such as a protein chain, can have some state variables (color, name, etc.) independent of the state of the child nodes. Therefore, the composite objects in our implementation are more complex than the composite design pattern [4] requires.

In our implementation we represent biochemical entities as objects and store all objects in a hierarchical tree structure (Figure 2). The levels of hierarchy are a convention that is not required by the data structure nor by the algorithms, but it simplifies both the implementation and the use of the tree. While Figure 2 shows only an example of the main hierarchy, the tree also contains relationships between physical entities, for example bonds and alignments, as well as other objects such as grid maps, surfaces, and arbitrary geometrical shapes. A grid is an array that contains discrete values sampled from a three-dimensional volume. Electron densities, force fields and spatial probabilities are types of data typically stored as a discrete grid. The density within a three-dimensional volume is visualized by an iso-surface, similarly to the way a contour curve indicates a specific height in a two-dimensional map. The coordinates of iso-surface points are computed from grid points by interpolation of density values. The surface points can be used to draw triangles, which approximate the iso-surface. We store the set of triangles as a separate surface object, which is a child of the grid object.

A bond between two atoms or an alignment between two or more protein chains is an object, although it represents a less tangible relationship between the entities. A normal object, for example an amino acid, is a branch node in a tree: a child of a chain and the parent of the atoms. A bond, however, does not fit into the tree as easily: the bond must connect two atoms without being either the parent or the child, since the presence of more than one bond per atom would violate the tree structure where each child can only have a single parent. Therefore, it is clear that a tree structure (Figure 3a) cannot be used to represent such relationships. Instead, we have used a directed acyclic graph (DAG) to describe both the molecular entities and their relationships (Figure 3b). This graph includes a tree, which is only a special form or subset of a DAG where each node has only one incoming edge; a tree does have the advantage of allowing more simplified recursive operations than a generic DAG. Consequently, we identify the tree within the DAG, marking the edges in the data structure to be of either a 'tree' or a 'relationship' type (Figure 3b). The edges belonging to the tree are stored separately from the relationship edges. Thus, we can access the data objects efficiently with tree algorithms, but the graph can still be traversed explicitly using the relationship edges when needed.

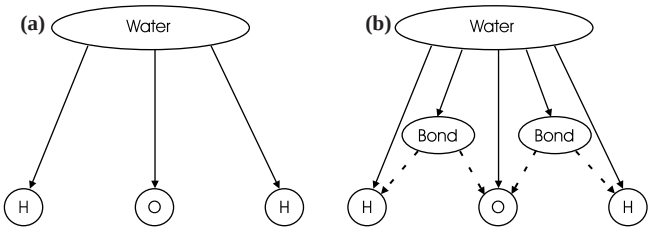

Figure 3. (a) Directed acyclic graph representing a water molecule. (b) Modified graph that contains a proper tree as a subgraph. The edges that do not belong to the tree are drawn with dotted lines.

A set of properties is implemented for each object type. Such properties include name, color, position, and selection. The position of an atom – essentially a point – contains two components: the initial *xyz* coordinates and the transformation, i.e. the rotations and translations that have so far been applied to that atom. The position of objects that are not leaves is either based on the position of their leaf objects, for example an amino acid residue has a position that is equal to the position of the C^{α}-atom, or the object does not have an implicit position, as is the case of a residue without defined atoms.

The most important tool for interactive data manipulation is the selection property, which has three states: 'selected', 'partial selection', and 'unselected'. These correspond to all, some, and none of the leaves of a node in the tree to be selected. Most algorithms and operations that manipulate data objects operate exclusively on selected objects. The 'partial selection' property enables the efficient search of the data tree to locate selected objects. Thus, the user can select a set of molecules using any of the graphical tools and then the computational algorithms will operate on that set of molecules. The selected and partially selected objects are highlighted with green and dark green colors, respectively, in order to make it easier for the user to identify the selected objects from graphical representations, such as from a list of objects, from an alignment, or from the three-dimensional display of the structure of the molecule. For example, the selection of a residue in the alignment will also immediately update the graphical view in order to highlight the position of the selected residue within the protein structure (Figure 4).

Implementation

The program has been implemented using the C++ programming language, although some subroutines in the modules have been obtained from existing C programs and have not been converted to C++: program components written with C can be directly called by the C++ main program. Despite the high level of abstraction, which allows the program design to closely resemble the modeled (biochemical) problem, the executable program produced from the C++ source code is efficient. There are also several well-designed C++ libraries available for all desired platforms. Consequently, the application developer can focus on the biochemical problem by using predefined program components from a library rather than wasting a large amount of effort on platform-specific implementation issues.

The high quality three-dimensional graphics uses standardized OpenGL [5]. There are OpenGL implementations for all common operating systems and graphics hardware. The graphical user interface was implemented using the Qt-library [6] because it provided a well-defined, multi-platform graphical framework. The Qt-library is a versatile framework, now supports the use of OpenGL graphics, and is available for platforms based on X11 (UNIX, Linux), Microsoft Windows, and Mac OS X. All graphical user interface objects – dialogs and menus – have been implemented using Qt. However, we have tried to minimize the Qt-dependencies of the main data structure interface, since the development of computational modules should be possible even in the absence of the Qt-library.

We began the development of Bodil on the SGI IRIX operating system, since at that time it was the dominant graphics workstation platform used for structural studies. Already then Linux-based PC machines were considered an important target platform, both due to the price of the hardware and software and because of the potential for future development. The graphics capability of the PC machines has now reached an impressive level with

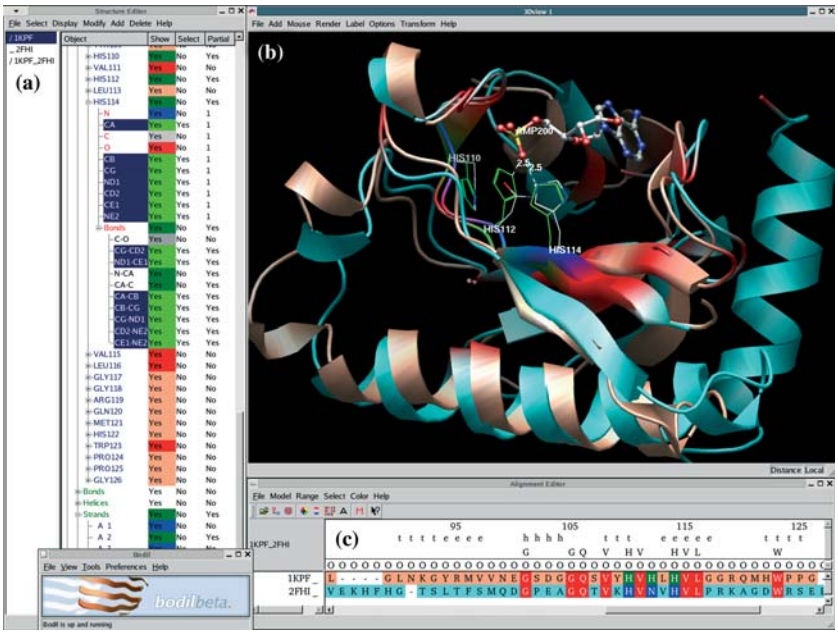

Figure 4. Three representations of two histidine triad proteins. (a) Hierarchical directory for data object management. The left pane lists the top level objects: protein kinase C inhibitor 1 (PDB code 1KPF [34]), fragile histidine triad protein mutant (PDB code 2FHI [35]; the second histidine of the histidine triad has been mutated to asparagine), and a structural alignment of the proteins (1KPF_FHI). The right pane in (a) shows part of the data tree for 1KPF with one residue (His114) expanded to show all the atoms. The side-chain atoms of the histidine triad are selected (blue and green) with related objects partially selected (dark green). (b) Three-dimensional graphics for structure visualization. The view focuses on the nucleotide binding site. Both 1KPF (gold) and 2FHI (light blue; conserved residues red on both) are drawn with ribbon and coil to illustrate the secondary structures. The AMP molecule bound to 1KPF is shown as a ball-and-stick model, and the side chains of the histidine triad of both proteins are shown as wire-frames (1KPF green, 2FHI in default atom color scheme). Distances between the phosphate O3P oxygen of AMP and the nearest atoms in 1KPF, the ϵ-nitrogen of His112 and the δ-nitrogen of His114, are visualized with white dashed lines. (c) Alignment view. The sequences of 1KPF and 2FHI are shown as aligned by structural superimposition. The color scheme for amino acids is the same as in (a) and (b): 1KPF gold, 2FHI light blue, conserved residues red, histidine triad dark green and blue, respectively. The residue numbering, secondary structure elements, and mutually conserved residues are shown above the alignment.

an affordable price. From the point-of-view of the programmer, IRIX and Linux (as well as other varieties of UNIX) are very similar platforms, and it has been a great asset to develop the program using both environments, since the different development tools (e.g., compilers, debuggers) available under the different operating systems complement each other. The Windows version proved to be more difficult to produce, mostly due to lack of experience with that platform and because Windows employs a different strategy for memory management. In Windows, the libraries are more self-consistent, with their own memory space, unlike Unix/Linux-based systems in which the allocated memory is owned by the program and not by the library. This caused extra difficulties, since in Bodil the memory allocated for data objects, which are usually created by parser modules or computational routines, must come from the memory address space of the main program. Since each module has its own address space in the Windows memory model, care must be taken that the memory allocation routines called from a module do use the memory pool of the main program rather than the memory local to the module. Even standard C++ container types – the dynamic and generic implementations of well-known data structures such as arrays, linked lists, and hash tables – cannot easily be accessed across the memory address space boundaries of the dynamic libraries in Windows. We are currently producing an Apple Mac OS X version of Bodil.

Results and discussion

The core program

Since the functionality of the program has been separated into individual modules, the main task

of the program is to hold data. Besides biochemical data, the core program also handles program settings. The main program reads all configuration files, including module-specific files, on startup. These data are used to generate the list of available modules without loading the binary library files for each module, which are loaded on demand. The module configuration may require a menu entry, which will be added at this point into the menu of the main program window. Access to menu items from the main window will load the corresponding modules. The module loading and execution interface is all that the core program knows about the modules, and therefore it is easy to add more functionality to the program package. The main program provides an application programming interface (API) for the modules to access the data tree and the configuration settings. The API includes an event notification system based on observer design pattern [4]. The observer pattern defines a one-to-many dependency, where the change of the state of the observed object (called 'Subject' or 'Publisher') is automatically signaled to each observing object (known as 'Observer' or 'Subscriber'). In practice, the subject has a list of observer objects and after each change of the subject state the list will be iterated through, and an 'update' procedure is executed for each list member. The observer object provides the 'update' procedure which changes the internal state of that observer to match the changes of the 'Subject' state. The main program has a single 'Subject' object that informs the observers about changes of the data tree. Any module may contain one or more 'Observer' objects that should implement appropriate actions for notification events. Thus, the modules can update their local data to correspond to the current state of the main data tree. For example, selecting a residue in the alignment view will be immediately visible also in the 3-D graphics and *vice versa*.

Description of available modules

The different modules can be divided into three functionally different categories: computational, graphical, and utilitarian modules. The utilitarian modules are used for viewing and changing the state of the program. For example, the configuration settings can be listed and edited in a separate window, which also allows the re-reading of the settings from files. Some modules do, however, cache the setting values and do not check if the values change. Most of those modules were developed relatively early in the project, when the core application was not yet available. Now that the main program design and implementation has reached a stable state, we are – besides adding functionality – revising older modules in order to replace redundant or more restricted initial implementations with calls to more refined implementations.

Computational modules

The computational modules contain algorithms for processing and producing molecular data. Many of the algorithms have been adapted from existing programs that were freely available, resulting in faster implementation and the introduction of fewer errors. Some of the algorithms were adapted from previously unpublished in-house software and are described here. When the main program and the primary modules reached a stable and usable state, we began to develop new algorithms and modules in order to expand the functionality of the program to better match the needs of molecular modeling as well as to test the robustness of the core design. As a result, the data structure design has been refined and augmented from its original model to the one presented here in order to better fulfill the needs of the computational algorithms as well as to improve overall performance.

Sequence alignments

The sequence alignment module was derived from the multiple sequence alignment program MALIGN [7]. The original command line program was implemented in C and used static memory management. The integrated version added a graphical dialog for selecting parameters, i.e., a scoring matrix and gap penalty, and the memory is now managed dynamically. While the use of dynamic memory handling is unavoidably slower than the use of static memory, it conveys two benefits: the memory used is only that which is really required, possibly compensating for any loss in performance, and the artificial problem size limit is removed. The module receives an alignment data object from the main program and recomputes the alignment of the sequences held by that object. Furthermore, the user may specify that some of the sequences belong

to a pre-aligned set, and the rest of the sequences are then aligned against that set and each other. A direct application of this latter feature is to align a sequence against a structural alignment made by comparing the three-dimensional structures of related proteins. Here, the structural alignment should have fewer ambiguities, but the sequence alignment algorithm is unlikely to reproduce it. By preventing changes to the structural alignment, we in effect introduce spatial variation into the gap penalties and bring structural information into the alignment process, leading to more accurate alignments.

Distances reflecting the degree of sequence similarity among proteins can be computed from the alignment and the relationships among sequences can be visualized with a tree diagram. For example, the Phylip program package [8] provides several methods for the distance computation and for phylogenetic tree construction. We have implemented as a separate module a graphical dialog where the user can select options, and the dialog will then produce and execute a script, which calls appropriate programs from the Phylip package in order to construct a tree from an alignment (currently available only in the Linux and IRIX versions).

Structural alignments

Proteins whose three-dimensional structures are known can be superimposed in order to compare their structures and to provide more accurate sequence alignments. We have added a separate module, Superimposer, which contains algorithms for superimposing protein structures. The pairwise superimposition of structures involves three steps: (1) the definition of equivalent positions in the two structures, (2) the calculation of the transformation that minimizes the distances between the equivalent positions, and (3) application of the transformation to the *xyz*-coordinates of one of the proteins so that it is superposed on the other. The transformation is computed with the least squares minimization method of Kearsley [9] and the transform is applied by adding it to the internal transformation stored within the atom objects.

The key step in the superimposition procedure is the definition of equivalences: the equivalences represent the common elements of the two compared structures. Thus, an incorrect set of equivalences usually produces a superimposition that will not correctly highlight the similarities and differences of the structures. At least three pairs of equivalent points are required in order to compute the transformation, and already three well chosen pairs can produce a useful superimposition. We provide three strategies for the definition of the equivalences: (1) manual method, (2) iterative superimposition-based procedure, and (3) fully automatic structure alignment.

In the manual method pairs of equivalences are defined by the user. The set of equivalent pairs should contain at least three pairs of atoms that are known or highly likely to reflect 'equivalent' entities in the structures located at the same or similar relative positions. They are often obtained from a sequence alignment or other data, for example mutation studies. The procedure obtains the equivalences from the specified alignment object, and minimizes the root of the mean squared deviations (RMSD) of the pairs of equivalent points, which by default are the C^{α}-atoms.

An approximate superimposition, obtained for example by interactively moving the molecules or by the use of the manual method, can be refined with the iterative method, which uses a dynamic programming algorithm (DPA) [10] for identifying the equivalences from superimposed structures. The similarity of the C^{α}-atom positions needed in the DPA is computed with the Rossmann and Argos [11] equation, similarly to the program STAMP [12]. The subset of identified equivalences that are already superimposed within a user defined cut-off are used to recompute the superimposition and the process is repeated until the set of equivalences does not change any more, or a maximum number of iterations is reached, indicating that the procedure cannot converge, but instead fluctuates between solutions.

The automatic algorithm implements our topology based structure alignment program VERTAA [13]. The algorithm was specifically designed to align structures that have similar shapes both quickly and automatically. VERTAA defines the initial set of equivalences by correlating the patterns of C^{α}-atom density values – the number of C^{α}-atoms within 14 Å of each residue – along the sequence of each structure. Therefore, no initial alignment or superimposition is needed. For convenience, multiple structures can be superimposed (pairwise) onto one reference structure with a single procedure call.

Molecular modeling of protein structure – Homodge
The prediction of a protein structure is a complex task, but it can be simplified if the complete answer is not needed or if enough biological information is available. For example, the specificity of a receptor can be studied quite extensively already when a model of the binding site, not the whole protein, is constructed. Homology modeling generates the model of a protein structure from known structures of homologous proteins (templates). Thus, both the selection of the template and the alignment of sequences have a significant effect on the model. If the model can be generated and displayed within seconds, the user can quickly assess the quality of the model based on his knowledge about the studied problem and decide to refine the current model, or feed information back to the alignment process and generate a new model.

The computer program Homodge has been developed for such use. The philosophy behind the program has been to rely on information from the related structure as much as possible, making the minimum number of changes. Such an approximation is sufficient when highly similar proteins are modeled and the speed of the method allows interactive use. While Homodge is particularly suitable for highly similar proteins, it also performs reasonably well with low similarity, unless the alignment contains long insertions or deletions (indels), since indels require construction, or at least major modification of the main chain.

Homodge generates the putative main chain of the model by copying the coordinates of all residues, including those that will be deleted, from the template structure. For each insertion the main-chain positions are computed by adjusting – with a simulated annealing algorithm – the main-chain torsion angles of the residues in the inserted fragment as well as for a couple of residues preceding and following the insertion. The goal of the torsion angle adjustment is to produce a continuous main chain with the torsion angles within allowed regions of the Ramachandran plot. Deletions are removed from the model after the insertions have been added, and torsion angles are adjusted in order to reconnect the remaining residues. This approach is fast for short insertions and deletions. The construction of the main chain is completed by adding atomic coordinates for the amino- and carboxy-terminal overhangs; both are given a pseudo-helical conformation. The residues are added in order to let the user manipulate the whole structure with other methods.

The resulting model is thus a partial copy of the template. There is no minimization step, so the side chains and even the main chain may collide. More accurate results would require minimization of the model structure, which is computationally expensive, and can itself produce structural artifacts. In places where the template differs only by point mutations (no indels) the main chain will be identical in the model regardless of the level of sequence identity.

When the main-chain coordinates have been determined, the side chains are reconstructed based on torsion angles, along with bond lengths and angles. The template provides torsion angles for the conserved residues, while the most frequently occurring torsion angles of each amino acid type – as found in known structures [14] – are used for inserted and mutated residues. The rotamer library (see below) can be used to quickly change the side-chain orientations.

The program Homodge is distributed as a separate binary usable either from the command line or from the alignment editor in Bodil. Thus, the user can also substitute an alternative modeling program for Homodge. For example, a shell script could execute Modeller [15] with predefined options and then return the produced model structure for Bodil.

Rotamer library
The conformation of side chains along the protein chain is important both for the structure of the protein and for the interactions with other molecules. We have added a rotamer library in order to facilitate the exploration of probable side-chain conformations. The rotamer library [14, 16] consists of a list of conformations corresponding to local energy minima that have been observed in known protein structures. The rotamer module within Bodil looks up the conformations for an amino acid from a list of identified rotamers [14] and creates an alternate set of coordinates for the atoms in the amino acid for each conformation. The user can cycle through the conformations for atoms, groups, or even whole proteins (NMR-structure files often report tens of alternative conformations for the whole structure, which can be cycled through with the same method as for rotamers of a single amino acid), and see on the

graphics how different conformations fit into the structure. The coordinates of any and all conformations can be saved to a structure file. The supplied rotamer library is a flat file in pseudo PDB-format [17] and can be edited or replaced by the user.

Surface computations

The space filling representation of atoms is an easily implemented way of showing the space occupied by a molecule, but the molecular surface is aesthetically and functionally a more effective method. The computation of the molecular surface is a three-stage process. First, a function is evaluated at discrete points. The function defines the shape of the surface. The evaluation of the function at any given point requires computing the distances from all atoms. Therefore, it is more effective to pre-compute values of grid points rather than to start from any point on the surface and dynamically explore the surface and use recursion to achieve an increased level of detail where the curvature of the surface is higher. A straightforward grid-based approach uses an even distribution of points resulting in evenly sized triangles, i.e. large planar surfaces will be divided into many triangles and tight curves flatten out due to too few triangles. A finer grid with more points does produce a more accurate surface composed of smaller triangles, but the memory requirements and amount of computations increase exponentially. The user can specify the spacing of the points in the grid in order to allow computation of both rough and smooth surfaces. A more complex, grid-based procedure would start with a sparse grid and recursively subdivide the volume locally into finer grids only where the surface is not planar. The recursive approach is not used in Bodil, since the grid data object type was designed for evenly spaced data points. A separate grid implementation for sparse data can be added later.

In addition to the molecular surface algorithm of Connolly [18], we have functions that will produce van der Waals and solvent accessible surfaces. The solvent is represented by a spherical probe with radius usually equal to 1.4 Å. The solvent accessible surface represents the closest points outside the molecule, where the probe can be positioned, i.e. positions where the distance between the center of the probe and the closest atom is equal to the sum of the probe radius and the van der Waals radius of the atom. When the probe 'rolls' over the molecule, its center moves on the solvent accessible surface and the probe touches the Connolly molecular surface. Thus, the Connolly molecular surface contains the volume within van der Waals radii of atoms and the grooves between two or more atoms that are unreachable by the probe.

The second stage is to create triangles such that they approximate a surface that represents a user-specified threshold value of the function. The position of each triangle is linearly interpolated from the grid points, since the value of the function is only evaluated for the grid points. We use the triangulation algorithm of Bloomenthal [19] to generate the triangles. The triangulation algorithm may be used to create a contour surface to any grid. For example, the user can read into the program an electron density map created from cryo-electron microscopy (cryo-EM) image processing and visualize the density volumes due to macromolecules.

The third stage in the surface computation involves locating the nearest atom for each point on the surface. The mapping of atoms to the surface allows the quick mapping of atom or residue properties such as color to the surface.

The initial implementation of the surface computation module performed its task as a single step and produced only the final surface object. We quickly found that the division of the process into separate sub-routines and the storage of the intermediate results gave much more flexibility and also helped us to verify the correctness of each sub-routine. Separating the computation of a grid from the generation of the iso-surface permits the generation of surfaces for precomputed grids. The creation of grid as a separate object allows the generation of several different iso-surfaces from one grid without re-computation of that grid. Different functions can be implemented for computing values to an existing grid. Thus, the user can choose from several options to perform different, but related tasks with a small number of program components. The default options produce basically the same surface as the initial single-step implementation did.

Electrostatic potentials

Molecular interactions are due to inter-atomic forces. The electrostatic potential is one important

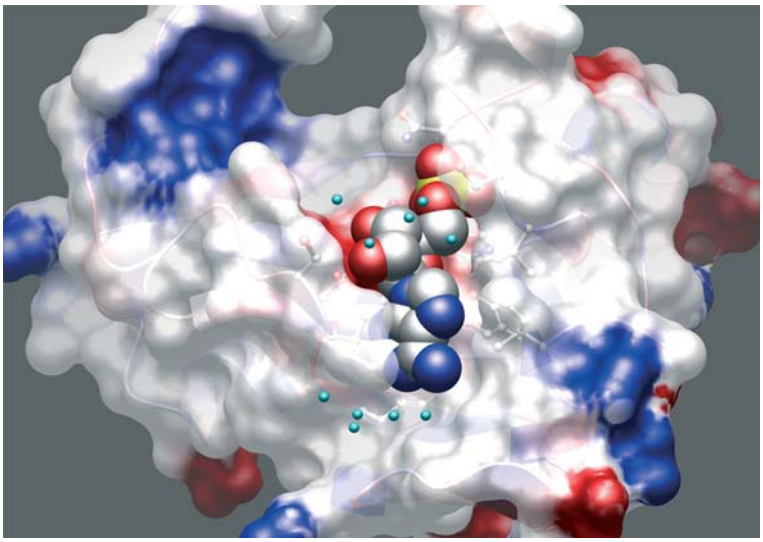

Figure 5. Molecular surface of protein kinase C inhibitor 1 (PDB code 1KPF [34]). The ligand molecule (AMP) is drawn as a CPK model. The secondary structure of 1KPF is shown. Residues with at least one atom within 4.0 Å from any atom in the ligand are shown as ball-and-stick representation. Oxygen atoms from water molecules within 5.0 Å of the ligand atoms are shown as small cyan spheres. The molecular surface for 1KPF has been computed, and the electrostatic potential has been interpolated onto the surface. The surface has been colored according to the potential values, from negative (solid red) through neutral (transparent white) to positive (solid blue).

type of force contributing to specificity in molecular interactions. The contribution of a charged atom to the potential energy falls off exponentially as the distance from the atom increases. However, the net potential energy at any specific point is a sum of contributions of all atoms, including solvent atoms. Visualization of electrostatic potentials for a protein is typically used to help locate the patches on the protein surface that are complementary to the surface of a molecule that interacts with the protein. For example, a surface in the binding pocket of a ligand molecule can have both hydrophobic (neutral) areas that provide non-specific interactions as well as charged/polar areas that interact specifically with charged/polar groups of the ligand. The computation of the electrostatic potential involves two steps: the potential is computed at discrete points (a grid) enclosing the protein, and then the grid values are visualized. Our implementation uses successive over-relaxation [20] to solve the Poisson–Boltzmann equation at the grid points. The visualization is handled by interpolating the values from the grid to points on the molecular surface computed for the protein. The surface is then colored based on these values, the value range is converted into a color range; the default color range is from red (negative charge) via semi-transparent white (neutral) to blue (positive charge) (Figure 5).

Fitting of atom-resolution protein models to low-resolution density maps

Large biomolecular complexes are increasingly studied by methods such as cryo-electron microscopy (cryo-EM) that provide three-dimensional, low-resolution density maps. Such a map can be used to construct a model of the complex at pseudo-atomic resolution, if the high-resolution X-ray or NMR structures of individual components are known or can be modeled from related structures. Thus, the task is to assemble the complex of molecules with the low-resolution density map as a guide.

We have implemented in the module 'EMfit' a simple algorithm to fit a protein structure into a density grid. The program performs an exhaustive search of the local rotational and translational space. Each sampled transformation is (i) scored according to the fit to the low-resolution density map and (ii) optionally examined for overlap with other protein structures contributing to the complex, which have been pre-determined by the user as being correctly positioned in the low-density map. The score is the sum of density values in the

low-resolution map at each C^{α}-position of the protein being fitted. Since side-chain atoms are not considered, possible inaccuracies in the protein model due to side-chain conformational changes upon complex formation have no effect on the fitting. Notably, the method uses neither predetermined density contour cut-offs to limit fitting nor convolution of the high-resolution model to the low resolution of the map. Density maps of entire complexes as well as difference maps, where the density of the components not being fitted has been deducted, can be used.

Transformations with overlap of protein structures can be discarded without further computations, since the non-overlapping solutions are guaranteed to be found by the exhaustive search. The overlap can be determined in linear time by the use of a simplified density grid, generated from the static structure. The transformed molecule overlaps if any of its atoms is within the density of the static molecule, i.e. a grid point near the atom has been flagged as being near a static atom (in practice, C^{α}-atoms closer than 4.0 Å from each other). Moreover, preferred positions for selected C^{α}-atoms can be pre-defined. The distances of these C^{α}-atoms from their preferred coordinates are then recorded in the list of transformations and they can be used for filtering the results. This allows external information about the location of the protein, such as mutagenesis or cross-linking data, to be included in the search. The user interface of the module allows the user to interactively specify the input parameters. After fitting, the user can pick transformations from the resultant list and immediately visualize the corresponding complexes.

The module has been successfully used to fit a receptor domain, the I-domain of $\alpha_2\beta_1$-integrin, onto the structure of the non-enveloped picornavirus echovirus 1 [21] (Figure 6) and to resolve the biological dimerization mode of bovine lysosomal α-mannosidase [22]. In the virus-receptor case, a density map of the complex at 25 Å resolution from cryo-EM and crystal structures of both echovirus 1 [23] and the α_2I-domain were available. Published results from mutagenesis were not used to constrain the fitting, but the resulting model was nonetheless in good agreement with those data [21]. Crystallographic data on α-mannosidase, an enzyme known to form a dimer in solution, was compatible with two alternative dimeric structures. The dimer was observed directly in solution by cryo-EM, and fitting of both of the dimers derived from crystal symmetry to the low-resolution cryo-EM density map clearly resolved the higher-scoring dimer as the biologically relevant one [22].

Parsers

Molecular data are usually stored in formatted files and there are several file formats, each designed for a different purpose. Most formats use text files that can be directly edited by hand. Such manual changes are convenient, but may introduce errors. Binary files are a more efficient form of storage, but are usually only accessible by a limited set of programs. Because we decided it would be important to be able to use and also produce files that are directly exchangeable and usable with other programs, Bodil does not use any Bodil-specific format: parser modules were developed where each module encapsulates knowledge about one existing, widely used format. Thus, each module is able to convert data from one format into data objects within Bodil and also to produce formatted data from data objects. The conversion from any particular file format has to take into account the conventions and peculiarities of that format. For example, the Protein Data Bank (PDB) [17] format assumes implicitly that the amino acids have bonds, while the SYBYL Mol2 [24] format lists each bond explicitly. Thus, some formats require more preprocessing and knowledge about represented molecules than others. In addition, the text files may be incomplete or modified by the user. In such a case the parser must be able to handle the inconsistent data gracefully. Since each format, including the data representation within Bodil, typically only represents some subset of the molecular data, the conversion has to use some default values for data that does not exist in the source format and to discard data that cannot be represented by the destination format. For example, the PDB format – designed for macromolecules – will not store the bond types for small molecules. Thus, the bond types for a small molecule should be computed, as they cannot be read from the PDB file. The current set of parser modules reads and writes the typical molecular formats PDB, Mol2, Gromacs [25], and MDL SDfile [26], and the sequence formats FASTA [27] and PIR. The modified PIR sequence

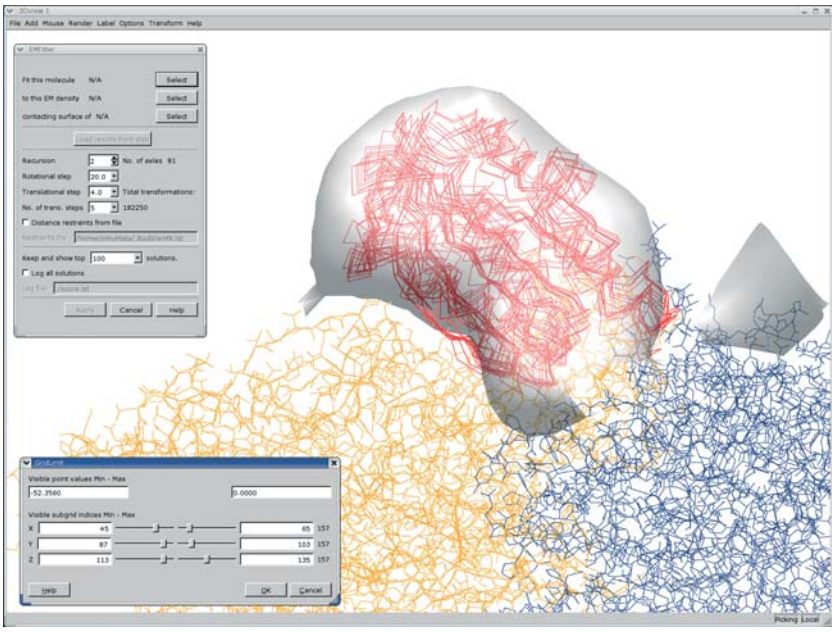

Figure 6. Rigid-body fitting of integrin α_2I domain into a low-resolution density map of the complex of echovirus 1 and α_2I domain using the EMFitter module [21]. The density map (isodensity contour surface shown in semitransparent gray) was obtained by cryo-electron microscopy of single particles and three-dimensional image reconstruction. C^{α}-traces of the 10 highest-scoring transformations of the α2I domain are shown in red. The atom-resolution crystal structure of echovirus 1 [23] was used to constrain the fitting of the receptor. Two protomers of echovirus are drawn in yellow and blue. The EMFitter start dialog and the interactive density map cropping dialog are shown on the left.

format is used also by the program Modeller [15], and we use it to represent sequence alignments too.

Another group of parsers is focused on importing three-dimensional value grids. These grids can represent, for example, electron densities or energy distributions. Thus, precomputed spatial data can be read in from ASCII grid formats produced by the programs Grid [3], AutoDock [2], CNSsolve [28], and Surfnet [29], and from binary formats of Grid and AVS [30]. Grids computed in Bodil can be written out for visualization by other programs. We are developing a molecular interaction-predicting algorithm using statistical methods [1, 31, 32]. The gathered data can be plotted to a grid and visualized just like any other spatial data. In Bodil, a grid can be easily visualized by computing a contour surface. The triangles of a surface, as well as spheres and cylinders representing atoms and bonds, can be saved as a Raster3D [33] input file for higher quality rendered images, but the 3D-graphics module can also print the rendered view directly into bitmap image. Likewise, the alignment viewer module can print the displayed alignment.

The C++ programming language encourages resource abstraction with the concept of the 'stream'. Both text and files can be referred to as streams of characters, thus separating the access to a stream from the true type of the stream. For parsing formatted data this creates an interesting possibility, since several file formats are text based and as such can be handled as streams. Thus, we have already implemented some of the parsers as procedures, which take data from, or put to, a stream, and then added a helper method that creates a stream from a file and calls the parsing procedure. The direct benefit from this separation is the possibility to call the parser module with a text stream object rather than a real file. To use this feature, we have a separate graphical module, which contains a box with editable text. Instead of saving data from Bodil into a text file and opening that file in a text editor, the user can display the same formatted text directly within the box. Likewise, the contents of the text box can be read into Bodil just like a regular data file, although the format of the contents must be specified by the user since there is no associated filename that

could indicate the correct format. Since the contents of the box can be edited, for example by pasting a sequence into the box, or by changing data retrieved from the Bodil data structure, the user can easily input data from several sources or even create several copies of a single molecule.

Graphical modules

The main purpose of Bodil is to visualize molecular data and provide the user with ways to modify that data interactively. The computational modules described above usually perform these modifications of data.

Structure editor

While the main program stores the data as a tree of objects (Figure 4a), the structure editor allows the user to explore the data object tree in a similar way to browsing directories and files. A file icon in a directory window shows some properties of the file and allows operations like rename and delete. Analogously, the structure editor lets the user see and change the properties of molecular objects. Naturally, the tree view allows selection and modification of individual objects as well as whole branches and groups of objects. Structure dependent selection methods have also been implemented. For example, all atoms near any already selected atom or within the same residue as the selected atoms can be easily selected. Thus, the user can quickly select a ligand molecule from the tree, request atoms within hydrogen bonding distance, and fully select each identified group. As a result, each protein residue within hydrogen bonding distance from the ligand is selected and thus easy to find both in the graphical view and in the sequence alignment. A separate query window is used to select atoms through a combination of three regular expressions that match chain, residue, and atom names, respectively. The query can select or deselect atoms matching the expression, or remove the selection if an atom does not match the expression. These operations correspond to union, difference, and intersection operators used in set theory. The structure editor also provides access to some computational algorithms: the calls to the surface computation module and the rotamer module require extra parameters (type of surface and target amino acid, respectively) that are supplied by the structure editor. The computational modules could – and should – have their own graphical dialogs for selecting such parameters, but it is more convenient for the user to access the modules via the structure editor and it reduces the number of windows displayed.

Three-dimensional graphics

The graphics viewer module (Figure 4b) can display one or more simultaneous 3D-graphical views, rendering each object in a different mode and from a different viewpoint. Multiple windows can be linked to receive the same camera rotation and translation changes. Most, but not all, objects do have a graphical representation. Atoms and bonds can be drawn with a combination of wire frame, stick, ball-and-stick, and CPK model representations. Atoms can also show their name, identifier (usually atom number from the structure coordinate file), charge, or type as a label. Similarly, groups can be labeled. A bond is drawn by default as two parts (lines or sticks) with each part using the color of the nearest atom, but it can also be constructed as a single piece with a unique color. The wire frame representation of an aromatic bond has one solid and one dashed line, while double and triple bonds have two and three solid lines, respectively. Texture bitmaps are used to draw aromatic bonds as 'stippled' sticks. The chain can be drawn as a wire frame C^{α}-trace or the secondary structure elements (SSE) can be drawn separately with helices as ribbons, strands as arrows of defined thickness, turns as filled arcs to highlight their locations, and random coils as tubes. The color of these chain fragments is either a single color of the particular SSE or the color assigned to the residues. The other shapes may have dot, line, opaque polygon, or transparent polygon representations. Surfaces have all four possibilities, while grids can only be drawn as dots. Only grid points with values within a user-specified range are drawn. Thus, 'dense' volumes can be located by specifying a range that covers the large values, but a contour surface computed for the grid, which encloses such volumes, is usually a more effective visual representation of the same property. Alignments of structures can be drawn as lines connecting the corresponding matched C^{α}-atoms, helping to highlight poorly superimposed, yet aligned residues. Furthermore, aligned residues can be colored according to the separation (distance) of their C^{α}-atoms in order to further highlight the differences

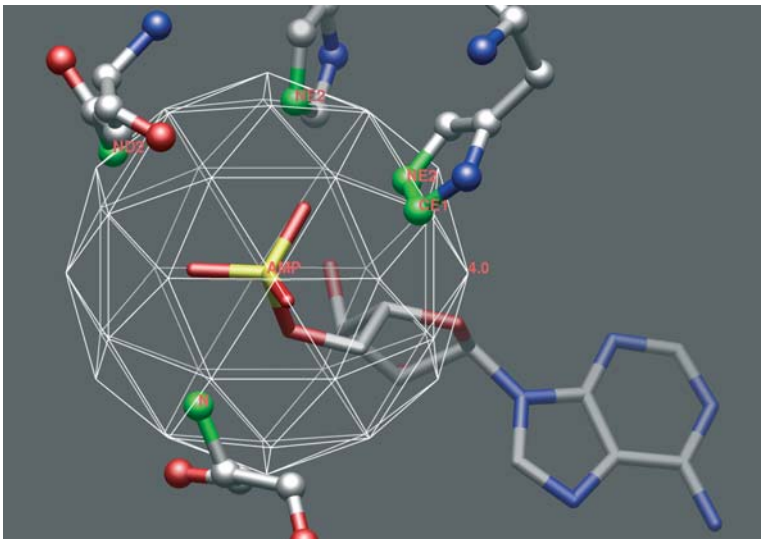

Figure 7. Selection sphere. The phosphate group of AMP bound to protein kinase C inhibitor 1 (PDB code 1KPF [34]) and the protein atoms around the phosphate group are shown as licorice and ball-and-stick models, respectively. The mouse movement 'drag' has been interpreted as the radius of a sphere that has been centered on the phosphorus atom (the start point of the 'drag' operation). The sphere is visualized dynamically with a wire-frame and the radius (4.0 Å) is shown. The sphere is used to select atoms within it: protein atoms within 4.0 Å from the phosphorus atom are green and have been labeled.

of the aligned structures. The view drawn on the screen exists within the memory of the graphics card. It can be saved as a bitmap image file, producing a snapshot of the view. However, that image is limited to the resolution of the visible graphics window. The OpenGL standard also describes off-screen rendering of the image directly into a bitmap format. This would allow the creation of higher-resolution images, limited only by the amount of available memory. However, the OpenGL implementations do not uniformly support the off-screen rendering to very large bitmaps. Therefore, only screen-resolution bitmap creation is available in the graphics module.

The mouse is used in several modes to produce different input actions. Movements of the mouse translate and rotate the viewpoint and change the zoom and slab, which determine the viewed volume. The coordinates of molecules can also be modified interactively with the mouse, whereby the rotation and translation change the atomic coordinates of the selected molecules. The mouse is also used to select atoms directly from the graphical view. Besides selecting single atoms, a sphere-selection mechanism is available: the user can select an atom and then drag the mouse to indicate the desired inclusion radius. The radius is visualized with a wireframe sphere (Figure 7); when the mouse button is released all atoms within the radius from the center point are selected.

There are other mouse selection modes that do not select atom objects but instead perform other actions. A centering function translates the camera to bring the pointed atom into focus. Measure modes add the pointed atom into a measure object. The measure objects are not common data objects; they are created and used only in the graphics module to compute and show either a distance, bond angle, or torsion angle for the atoms they refer to. Thus, selecting two atoms with the distance mode will add to the view a dashed line connecting those atoms and a number showing the distance. The displayed value changes automatically if the distance of the atoms changes, since it is computed on-the-fly. Since a distance object specifies two atoms, it can be used to create a bond between those atoms. Several options affecting the quality of the graphics can be changed interactively, like fog effects, ambient light, the anti-alias feature of lines and stereo parameters; even more options are available in the configuration file.

The graphics can also produce both wall-eyed and cross-eyed side-by-side stereo and hardware stereo views. The latter requires quad-buffered OpenGL support from the underlying graphics

system. Proper anti-aliased lines and transparent surfaces would require drawing the objects in back-to-front order, but the sorting of objects needed after each change of viewpoint is a time-consuming operation. In order to keep the graphics fast enough for interactive use, we have chosen not to sort the objects. Instead, we apply a 'peeling' method, which requires rendering of the view in several passes: (1) determine the closest objects, (2) render other than closest objects, and (3) render the closest objects. This ensures that at least the closest objects have proper anti-aliasing and transparency.

Alignment editor

The purpose of the alignment editor (Figure 4c) is to display protein sequences and to ease the editing of alignments. Since each character in the displayed sequence represents either a gap or a residue, and the linkage between a residue character and the corresponding residue object is maintained, the residue objects can be manipulated – for example, selected – via the alignment display. The primary visualization aid available through the alignment editor is the coloring of residues. Besides changing the color of selected residues, the user can color residues based on their properties. The properties for residues (e.g. hydrophobicity, hydrophilicity, polarity, etc.) are listed in a data file and a color map is associated with each property. Thus, the property values are converted into colors and applied to the residues. Since the properties are defined statically for individual amino acid types, the values and colors do not depend on the sequence. However, residues do have properties that are sequence or structure dependent – for example, the solvent accessible area of a residue, or the accuracy of the superimposition of the structurally aligned residues – which must be computed rather than looked up from a table. Thus, the generation of a property value, be it from a table lookup or computation, should be clearly separated from the visualization of the value, for example with color schemes or separate plots.

The main function of displaying an alignment is to identify and highlight similarities and differences among the sequences. Therefore, two coloring methods operate on the properties of the alignment, rather than properties of individual amino acids, as above. One identifies each column in the alignment where a user-specified minimum number of residues are identical and then colors those residues, highlighting conserved amino acids. The other alignment-based coloring method computes the distance between C^{α}-atom positions of aligned residues and converts the distance value into a color using a color map. However, the distance computation assumes that the structures have already been superimposed according to the alignment. The superimposed residues in the alignment will have different color (blue) from those residues (red) that, although aligned, have distance larger than a cut-off value (6.0 Å). Both the color map and the cut-off are adjustable configuration options.

The object shown in the alignment editor is an alignment object. It contains a list of chains and an array of columns, each representing one position in the alignment. A column in the alignment points to an aligned residue in each chain or shows a gap marker for those chains that do not have a residue corresponding to that position. The alignment objects are generated on three occasions: (1) from (unaligned) sequences in the alignment editor, (2) as a result of the structure alignment algorithm, or (3) from a file with pre-aligned sequences. It is important to note that the alignment object may contain any set of sequences, with gaps inserted at arbitrary positions. Only if the sequences do have corresponding parts – residues – and those residues are aligned properly, then the alignment object does represent an alignment of the protein sequences (or structures), else the alignment object merely lists the residues. Deletion of a residue object does not destroy the alignment; the deleted residues are converted into gap markers in the alignment.

The alignment objects are modified by passing them to the Malign module, which performs the sequence alignment, or by adding and removing columns or rows and by moving gaps in a row interactively. The structure alignment module creates new alignment objects in order to describe the alignment of structures instead of modifying existing alignments. The user may also specify a subrange of the sequences for re-alignment by Malign, for example to realign a loop region while keeping the rest of the alignment unchanged. It is possible to have more than one alignment editor window open at the same time in order to view several alignments or different locations along a

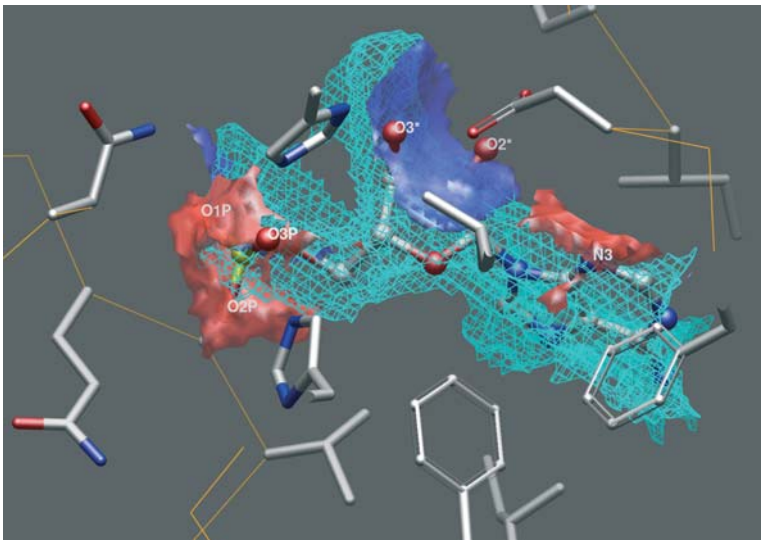

Figure 8. Visualization of precomputed spatial data. A molecular interaction library [32] has been used to predict volumes in contact with the protein kinase C inhibitor 1 structure (PDB code 1KPF [34]) where ligand atoms can be placed: hydrogen bond donors (blue), hydrogen bond acceptors (red), and aliphatic carbons (cyan wire-frame). The predicted volumes coincide closely with the ligand molecule (AMP, ball-and-sticks, six atoms labeled) present in the crystal structure. Amino acid side chains around the ligand have been drawn as licorice models and the C^{α}-trace (orange) is also shown.

single alignment that are difficult to visualize within a single window because of the length of the alignment.

Performance and system requirements

The size of the program is moderate, less than 100,000 lines of code, and the size of the binary distribution is about 20 MB, including the Qt-library binary file which is required by the program. The memory use of the program depends on the amount of objects (molecules, grids, surfaces) that are loaded or computed during runtime. The development of the program began on the IRIX platform using MIPS R5000-based SGI O^2 workstations. The main asset of those machines was their support for hardware stereo graphics. Interestingly, the graphics system in the O^2 workstation used the main memory rather than dedicated graphics memory, and performed part of the rendering in the main processor. When the PC graphics hardware evolved into a stereo-capable platform, Linux became the main development platform. While OpenGL provides a standard interface for graphics, the optimization of the graphical performance must take into account the hardware used. There is a clear trade-off between speed and quality. One way to increase speed without sacrificing quality is to minimize the computations made on the graphics hardware by precomputing values in the main processor. This, however, requires more memory for storing the values and the transfer of values from the main memory into the graphics memory may well be the real bottleneck of the system. In the old O^2 systems the memory was limited, and the computations were in every case performed by the main processor. On the latest PCs as well as on high-end SGI workstations the graphics hardware is very advanced and most likely contains more computational power than the main processor.

Planned additions

The program, although usable, is by no means complete yet. The basic modeling procedure would benefit from addition of more functionality. Furthermore, the feedback from users has not only helped to remove bugs, but has elongated the list of 'obvious' features needed, too. For example, full-featured command-line access and scripting, and saving of the program state would allow continuation of work at a later time. Direct access to data on network servers and databases may be more convenient than via intermediate files. The chemical properties of molecules obtained from a

database for a large set of molecules or from computations can be intractable without a tabular view that supports sorting and filtering of data. Similarly, both quantitative structure–activity relationships (QSAR) and small molecule docking are essential computational tools for drug design. Rantanen et al. [1, 31, 32] have studied molecular interactions detected in the experimentally determined structures of protein–ligand and protein–protein complexes. The resulting interaction library could, for example, be used to score how well a small molecule has been docked into the binding site of a protein (Figure 8). Additional interactive visualization methods, such as contact maps, Ramachandran plots, and trees, could highlight potential errors within structural models. Building of molecules, adjustment of torsion angles, and automatic addition of hydrogens and hydrogen bonds would contribute to the modeling process. The increase of configurability and addition of graphical options would allow production of even better quality images.

Conclusions

We have developed a modular program package, Bodil, for visualization of molecular structures and interactions in order to simplify the study of proteins and their function. The use of available program development tools and technologies such as the OpenGL standard and the Qt library has facilitated simultaneous development on different platforms, such as Linux and MS Windows. One of our goals was to produce a high quality program operating on PC and portable computers, eliminating the need for high-priced graphics workstations. The modular design of the program allows addition of functionality in the form of new modules without changing the existing program. It was important to implement the functionality as small specific algorithms that could be mixed and matched – reused – to offer a wide spectrum of possibilities to the user. We already provide the essential modules used in structural studies and protein homology modeling, and currently it is rather straightforward to introduce new ideas and functionality quickly.

The visualization of biochemical data offers several benefits. While an algorithm may yield an optimal result for a ligand docking with respect to the used scoring function, often only the human eye can deduce from a three-dimensional image whether the result is also biochemically sound. The ability to see data in several formats simultaneously allows the user to recognize such relations that cannot be seen in any single view alone. For example, seeing both a conserved residue in the sequence alignment and the position of that residue in the protein structure may help to explain why that residue is conserved. The interactive graphics allows the user to focus on the key features of the studied molecular interactions. Furthermore, the user can interactively select and limit important substructures as input for the computational algorithms and the result of the computation will automatically be visualized.

The beta version of the program is available for use without a fee. Compiled versions of the program for SGI IRIX, Linux, and MS Windows operating systems, as well as a tutorial, can be downloaded from: http://www.abo.fi/fak/mnf/bkf/research/johnson/bodil.html.

Acknowledgements

We are grateful for constructive feedback and encouragement from members of the Structural Bioinformatics Laboratory in the Department of Biochemistry and Pharmacy, Åbo Akademi University. This study has financially been supported by TEKES – the National Technology Agency of Finland, the Academy of Finland, the Ministry of Education, the Foundation of Åbo Akademi University, the Erna and Victor Hasselblad Foundation (Sweden), the Sigfrid Juselius Foundation, the Tor, Joe och Pentti Borgs minnesfund, Fatman Bioinformational Designs Ltd, the National Graduate School in Informational and Structural Biochemistry, Finland, and the Graduate School in Computational Biology, Bioinformatics and Biometry, Finland.

References

1. Rantanen, V.V., Denessiouk, K.A., Gyllenberg, M., Koski, T. and Johnson, M.S., J. Mol. Biol., 313 (2001) 197.
2. Morris, G., Goodsell, D., Halliday, R., Huey, R., Hart, W., Belew, R. and Olson, A., J. Comput. Chem., 19 (1998) 1639.

3. Goodford, P.J., J. Med. Chem., 28 (1985) 849.
4. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (Eds), Design Patterns. Elements of Reusable Object-oriented Software. Addison-Wesley, New York, USA, 1994.
5. Woo, M., Neider, J., Davis, T. and Shreiner, D., OpenGL Programming Guide, 3rd edition. Addison-Wesley, New York, USA, 1999.
6. Qt, Version 3.3. Trolltech AS, Oslo, Norway, 2004.
7. Johnson, M.S. and Overington, J.P., J. Mol. Biol., 233 (1993) 716.
8. PHYLIP (Phylogeny Inference Package), Version 3.5c. Joe Felsenstein, Department of Genetics, University of Washington, Seattle, 1993.
9. Kearsley, S., Acta Crystallogr., A, Found. Crystallogr., 45 (1989) 208.
10. Fredman, M., Bull. Math. Biol., 46 (1984) 553.
11. Rossmann, M. and Argos, P., J. Mol. Biol., 105 (1976) 75.
12. Russell, R.B. and Barton, G.J., Proteins, 14 (1992) 309.
13. Johnson, M.S. and Lehtonen, J.V., In Higgins, D. and Taylor, W. (Eds), Bioinformatics: Sequence, Structure and Databanks. Practical Approach Series, Oxford University Press, Oxford, UK, pp. 15–50.
14. Lovell, S.C., Word, J.M., Richardson, J.S. and Richardson, D.C., Proteins, 40 (2000) 389.
15. Šali, A. and Blundell, T.L., J. Mol. Biol., 234 (1993) 779.
16. Dunbrack, R.L. Jr. and Karplus, M., Nat. Struct. Biol., 1 (1994) 334.
17. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N. and Bourne, P.E., Nucleic Acids Res., 28 (2000) 235.
18. Connolly, M.L., Science, 221 (1983) 709.
19. Bloomenthal, J., In Heckbert, P.S. (Ed.), Graphics Gems IV. Academic Press Professional, Inc., San Diego, CA, USA, pp. 324–349.
20. Nicholls, A. and Honig, B., J. Comput. Chem., 12 (1991) 435.
21. Xing, L., Huhtala, M., Pietiäinen, V., Käpylä, J., Vuorinen, K., Marjomäki, V., Heino, J., Johnson, M.S., Hyypiä, T. and Cheng, R.H., J. Biol. Chem., 279 (2004) 11632.
22. Heikinheimo, P., Helland, R., Leiros, H.K.S., Leiros, I., Karlsen, S., Evjen, G., Ravelli, R., Schoehn, G., Ruigrok, R., Tollersrud, O.K., McSweeney, S. and Hough, E., J. Mol. Biol., 327 (2003) 631.
23. Filman, D.J., Wien, M.W., Cunningham, J.A., Bergelson, J.M. and Hogle, J.M., Acta Crystallogr. D, Biol. Crystallogr., 54 (1998) 1261.
24. SYBYL®, Version 6.7.1, Tripos Inc., St. Louis, Missouri, USA.
25. Lindahl, E., Hess, B. and van der Spoel, D., J. Mol. Model., 7 (2001) 306.
26. MDL ISI/Base, Version 2.5, MDL Information Systems Inc., San Leandro, CA, 2002.
27. Pearson, W.R. and Lipman, D.J., Proc. Natl. Acad. Sci. USA, 85 (1988) 2444.
28. Brünger, A.T., Adams, P.D., Clore, G.M., DeLano, W.L., Gros, P., Grosse-Kunstleve, R.W., Jiang, J.S., Kuszewski, J., Nilges, M., Pannu, N.S., Read, R.J., Rice, L.M., Simonson, T. and Warren, G.L., Acta Crystallogr. D, Biol. Crystallogr., 54 (1998) 905.
29. Laskowski, R.A., J. Mol. Graph., 13 (1995) 323.
30. Advanced Visual Systems Inc., Waltham, MA, Developer's Guide and Applications Guide.
31. Rantanen, V.V., Gyllenberg, M., Koski, T. and Johnson, M.S., Bioinformatics, 18 (2002) 1257.
32. Rantanen, V.V., Gyllenberg, M., Koski, T. and Johnson, M.S., J. Comput.-Aided Mol. Des., 17 (2003) 435.
33. Merritt, E.A. and Bacon, D.J., Meth. Enzymol., 277 (1997) 505.
34. Lima, C.D., Klein, M.G. and Hendrickson, W.A., Science, 278 (1997) 286.
35. Pace, H.C., Garrison, P.N., Robinson, A.K., Barnes, L.D., Draganescu, A., Rosler, A., Blackburn, G.M., Siprashvili, Z., Croce, C.M., Huebner, K. and Brenner, C., Proc. Natl. Acad. Sci. USA, 95 (1998) 5484.