BUTTERWORTH HEINEMANN

Biomolecular visualization using AVS

Bruce S. Duncan, Tom J. Macke, and Arthur J. Olson

The Scripps Research Institute, La Jolla, California

Dataflow systems for scientific visualization are becoming increasingly sophisticated in their architecture and functionality.[2–6] AVS, from Advanced Visual Systems Inc., is a powerful dataflow environment that has been applied to many computation and visualization tasks.[2,7–9] An important, yet complex, application area is molecular modeling and biomolecular visualization. Problems in biomolecular visualization tax the capability of dataflow systems because of the diversity of operations that are required and because many operations do not fit neatly into the dataflow paradigm. Here we describe visualization strategies and auxiliary programs developed to enhance the applicability of AVS for molecular modeling. Our visualization strategy is to use general-purpose AVS modules and a small number of chemistry-specific modules. We have developed methods to control AVS using AVS-tool, a programmable interface to the AVS Command Line Interpreter (CLI), and have also developed NAB, a C-like language for writing AVS modules that has extensions for operating on proteins and nucleic acids. This strategy provides a flexible and extensible framework for a wide variety of molecular modeling tasks.

Keywords: scientific visualization, AVS, molecular modeling

INTRODUCTION

Dataflow environments for scientific visualization are becoming increasingly popular. Dataflow systems include AVS,[2] IRIS Explorer,[3] Data Explorer,[4] Khoros,[5] and apE.[6] These environments were first developed around 1989, and since then, their architecture and capability have improved greatly.

Dataflow environments are an example of the technique of visual programming, in which independent building blocks (modules) are connected graphically to produce a network that implements a particular visualization or computation. An important feature of this strategy is that modules can be connected in different ways to perform different tasks.

Molecular modeling and biomolecular visualization pose many challenges for dataflow visualization environments. Molecular modeling uses many different types of data, such as atom coordinates, atom types, atom properties, bond connectivity, molecular surfaces, electron density maps, and electrostatic potential. Molecular data include information of various dimensions such as scalar (e.g., charge, hydrophobicity), vector (e.g., force fields, electric field, normal mode vectors), or tensor (e.g., anisotropic temperature factors). The spatial location of these data may be at isolated points, on a two-dimensional surface, or within a three-dimensional volume. Also, many of these data can change with time.

These diverse needs have been addressed by several commercial and academic molecular modeling packages. Most of these packages, however, are monolithic programs where the user has limited flexibility to alter the style of the visualization or extend the program to accept new types of data or do new computations. Since they do provide a good environment for many well-defined tasks in molecular modeling, they are appropriate for the routine needs of many computational scientists. However, part of the mission of the Olson laboratory at The Scripps Research Institute is to develop new methods for biomolecular visualization; by definition, new methods are not included in commercial systems.

Although there are tasks for which commercial modeling packages are clearly superior to AVS, many of our visualization tasks are effectively performed within the AVS environment. We use AVS for analyses that are not well suited to available programs. These include rapid prototyping, and visualization to support the development of molecular computation software. Several programs developed in the Olson laboratory (e.g., AutoDock,[10,11] Harmony,[12] SURFDOCK[13]) use AVS as an important tool to aid in software development. A key advantage of AVS for our work is that molecular analysis is performed in an environment that already has extensive tools for the analysis of three-dimensional data. Also, the rendering capability of AVS is of very high quality, thus allowing us to concentrate on other aspects of the visualization process.

Color Plate for this article is on page 299.

Address reprint requests to Dr. Olson at The Scripps Research Institute, 10666 North Torrey Pines Road, La Jolla, California 92037.

Received 5 June 1995; accepted 22 July 1995
This article is an updated version of the paper "AVS for Molecular Modeling," which was presented at the 1993 AVS Users Group Conference in Boston, Massachusetts (see Ref. 1).

655 Avenue of the Americas, New York, NY 10010

0263-7855/95/$10.00
SSDI 0263-7855(95)00067-G

Here, we present visualization strategies developed to support a wide-range of biomolecular visualization tasks. Additional chemistry applications of AVS have been described by others.[7,14,15] Space limitations do not permit us to describe completely the analysis of molecular three-dimensional fields using standard AVS modules, molecular animation techniques, visualization using texture mapping,[16] or new methods of normal mode analysis.[17] We also do not describe our work in volume rendering and other visualization techniques that could be incorporated into the AVS environment.[18,19] We only describe our experience with AVS version 5.02, and do not describe in detail features of the new AVS/Express environment. Our visualization tasks include the following:

- Display of molecular structure
- Analysis of molecular dynamics trajectories
- Presentation graphics
- Visualization of electron density maps
- Visualization of molecular affinity grids
- Protein–ligand docking
- Image processing
- Display of molecular surfaces
- Selection of molecular representation
- Normal mode analysis
- Analysis of Brownian dynamics
- Visualization of electrostatic potential
- Molecular animation
- Protein–protein docking
- Visualization of quantum calculations
- Analysis of protein folding

Several of these, for example the analysis of three-dimensional scalar fields, can be done with standard AVS modules, while others can be done only with custom AVS modules. We believe that the power of AVS is the availability of many general-purpose modules that can be applied to a wide variety of tasks. Our strategy continues this philosophy; we have developed many general-purpose modules that manipulate the types of data that we use. Chemistry-specific modules are developed as needed. We have also integrated entire chemistry applications into the AVS environment.

The work described here was done at The Scripps Research Institute by B. Duncan, M. Sanner, Y. Chen, and J. Yng in the laboratory of A. Olson; T. Macke and J. Smith in the laboratory of D. Case; A. Shaw in the laboratory of D. McRee; and M. Pique.

MOLECULAR DATA TYPES

Typical molecular data consist of atom coordinates and atom properties such as atom name, atom type, radius, color, charge, hydrophobicity, and bond connectivity. Molecular data also include two-dimensional surfaces and three-dimensional volumes.

An important concern when constructing any computer program is the data structures; this is also important in AVS. Instead of using chemistry-specific data types, we use standard AVS fields. This allows the use of many standard AVS modules such as READ FIELD, FIELD MATH, EXTRACT SCALAR, EXTRACT VECTOR, COLORIZER, PRINT FIELD, and WRITE FIELD.

Our modules use several types of input fields. Since many of the biochemical data types describe information at isolated points in space, we encode many of the data types as "uniform 1D *n*-vector real" fields. The semantics of the data, that is, the physical property that the data represents, is defined by the user. Table 1 shows our typical AVS data types.

Whereas there are conventions within AVS for describing the format and semantics of two-dimensional or three-dimensional scalar and vector fields, there are few conventions for describing molecular surfaces or scatter data independent of its geometric representation. One option is the unstructured cell data (UCD) data type. We chose not to use the UCD data type because its initial implementation did not have good performance, it is more complicated than the field data type, and there are few modules that manipulate UCD data. Several of these concerns have been addressed in recent releases of AVS, and we believe that the new facilities of AVS/Express will facilitate data management.

Molecular data, as with most complex data, come in many formats. For macromolecular computations, there are several "standard" data formats. Also, each institution or laboratory may have its own data formats. To read data into AVS, the user writes an AVS field header file that describes

Table 1. AVS data types for molecular data

Data type	Data description	AVS field type
$x\ y\ z$ coordinates	Atoms, surface points	uniform 1D 3-vector real
$x\ y\ z\ n_x\ n_y\ n_z$	Points with normals	uniform 1D 6-vector real
$x\ y\ z\ x'y'z'$ coordinate pairs	Vectors, disjoint lines	uniform 1D 6-vector real
$\Delta x\Delta y\Delta z$ coordinate offsets	Animation offsets	uniform 1D 3-vector real
$x\ y\ z$ coordinate time series	Animation coordinates	uniform 2D 3-vector real
αRGB	Atom or surface color	uniform 1D 4-vector byte
$(u,\ v)$	Texture map indices	uniform 1D 2-vector real
$q(x,y,z)$	Scalar property at a point	uniform 1D 1-vector real
$v(x,y,z)$	Vector property at a point	uniform 1D 3-vector real
$\rho(x,y,z)$	Three-dimensional scalar field	rectilinear 3D 1-vector real
i index field	Data selection	uniform 1D 1-vector integer
$i\ j$ connectivity	Bonds, disjoint lines	uniform 1D 2-vector integer
$i\ j\ k$ connectivity	Surface triangles	uniform 1D 3-vector integer

the content and format of the data file. In some cases, a custom AVS module is written to read the data, although we cast our data into the AVS field datatype whenever possible.

Typical molecular file formats consist of a collection of records given one per line. Each record consists of data items separated by spaces or tabs. We try to use this type of data file whenever possible, and avoid optional fields since it is impossible to read these data files as AVS fields; AVS fields cannot have optional elements (AVS/Express, however, does have conventions for null data).

Data files are usually represented as a "scatter" field, that is, a "uniform 1D *n*-vector" field. To aid in reading scatter fields into AVS, we have written MKAVSHEADER, an awk[20] script that reads an ASCII data file and writes the AVS header file. Columns of character data are ignored, and columns of constant data are ignored optionally. MKAVSHEADER has options for adding comments and labels, and the user name, date, and directory information are added automatically. Header files for two-dimensional or three-dimensional data are written manually, often from templates generated by MKAVSHEADER.

MKAVSHEADER is useful for creating field descriptions of the coordinate records of typical atom data files such as Brookhaven Protein Data Bank (PDB) files.[21] The atom coordinates of a PDB file often has a structure similar to the following:

```
ATOM      1  HN1  THR     1      17.017  14.972   4.068
ATOM      2  HN2  THR     1      16.297  13.912   2.883
ATOM      3  N    THR     1      16.982  14.095   3.587
ATOM      4  HN3  THR     1      17.707  14.470   3.008
ATOM      5  CA   THR     1      16.949  12.808   4.348
```

For this case, MKAVSHEADER creates a "uniform 1D 4-rector real" field of five elements with the four vector components defined by columns 2, 6, 7, and 8. (Column 5, which has the constant value of 1, is ignored by default.) One complication with the PDB format is the existence of optional fields and data elements that are not separated by spaces or tabs (fixed format records). MKAVSHEADER cannot process data files with optional fields or files with elements not separated by spaces or tabs. MKAVSHEADER can create header fields for data that can be represented as a one-dimensional scatter field. Examples include: atom coordinate files, atom property files, bond connectivity, vertices of molecular surfaces, spherical harmonic surfaces, and normal mode vectors.

AVS provides distinct mechanisms for storing the spatial coordinates and the data values for each data point. We chose to encode the coordinate information as part of the data so that standard AVS modules could operate on the coordinates and the data. This is particularly important because AVS does not provide mechanisms for performing operations on the coordinate information. We prefer a more uniform framework of treating coordinates and data in the same manner.

GEOMETRY CREATION MODULES

The translation of data to a geometric representation (spheres, lines, polygons, tubes, cones, ellipsoids) is a key element of the visualization process. Our typical geometric representations are shown in Table 2.

We generate these representations in a straightfoward manner, and have written modules that convert AVS field data that represent geometric information (e.g., vertices, radii, colors, triangle connectivity) into geometry objects. These modules provide the core functionality for creating complex visualizations.

For the modules described below, the input is the geometry specification represented as standard AVS fields, and the output is a geometry object that can be rendered by the AVS GEOMETRY VIEWER. Each module has required inputs, and usually has several parameters and optional inputs. For example, color information is typically optional. Another optional input is the vertex data field. This field, an integer value associated with each vertex, is typically used during picking.

SPHERES module

The SPHERES module creates a set of spheres using an input coordinate field and optional radius, color, and vertex data fields. If no radius field is given, all radii are controlled with a dial. The user can specify the name of the geometry object, disable the color field (so that the color of all the spheres can be set with the GEOMETRY VIEWER), and add pick information to each sphere. When using the optional color field, the SPHERES module can scale each sphere radius using the transparency (alpha) channel of the color field. Color fields are typically created using the AVS GENERATE COLORMAP and COLORIZER modules. The result is a field that contains red, green, blue, and transparency data. The transparency ranges from 0 to 255 and is scaled to a range of 0 to 1. The transparency value can be used to scale each sphere radius. By this method, we can change the transparency values of the colormap to interactively control the size of the spheres. This is useful to emphasize certain spheres and deemphasize others.

DISJLINES module

The DISJLINES module generates a set of disjoint lines (line segments) using a coordinate field and an index field. This module also accepts an optional color and vertex data field. There are several ways to specify disjoint lines, and for each method the second point of the line segment can be

Table 2. AVS geometric representations

Data type	Geometric represenation
Atom data	Colored spheres, lines, tubes, ellipsoids
Surface data	Colored triangles, spheres, wireframe mesh, texture maps
Volume data	Slice planes, isocontours, data sampled on arbitrary objects

interpreted as an absolute coordinate or as an offset from the first point. In one method, a six-vector representing initial and final points is given in a single data field. Another method uses two three-vector data fields representing the coordinate pairs. The third input method uses the index field that defines the line connectivity. This field is given by a "1D 2-vector integer" field where the components are the 1-based indices of the initial and final points in the coordinate field for each line segment. This is the method used to display the bonding structure of a molecule. The user can set the name of the object created, disable the color field, or scale the value of the second coordinate of the line segment (to adjust the length of normal vectors, for example).

POLYLINE module

The POLYLINE module accepts a single coordinate field and optional color and vertex data fields. It produces a single polyline that connects the input points. The user can set the name of the object created and disable the color field.

POLYGON module

The POLYGON module creates a geometry object composed of polygons, from a coordinate field and a 1-based index field that references the vertices of each polygon. This module accepts optional fields representing color, vertex data, polygon data, two-dimensional texture map indices, three-dimensional texture map indices, and vertex normals. The user can set the name of the object created and disable the color field. By default, POLYGON accepts fields that define triangles. Polygon can also draw arbitrary convex polygons (pentagons, hexagons, etc.) with the following caveat: the vector length of the index field that specifies the connectivity must be the size of the largest polygon, and entries for unused indices of polygons less than this size are set to -1.

POLYTUBE, ELBOW, and COLORTUBE modules

Another visualization technique is the tube representation. The POLYTUBE module creates a cylindrical object from a polyline. The radius and the number of polygons in the circular cross-section of the tube are controlled interactively. The ELBOW module controls the smoothness of the joints between the line segments. These modules were written by Y. Chen.

The COLORTUBE module is an extension of the AVS TUBE module. This module accepts a disjoint line geometry and produces a tube for each line segment. The user can specify the tube radius and the number of segments in the tube cross section. Also, this module can collapse each tube at either end point to produce a cone; this is useful for visualizing vector fields.

GENERAL-PURPOSE MODULES

In addition to the modules described above that create geometry objects from input fields, we have developed other general-purpose modules for tasks such as sending images to remote frame buffers, controlling videodisc recorders, performing math operations on parameter data, switching multiple input fields to output fields, clipping molecular surfaces, extracting data subsets using an index field, downsizing the triangulation of spherical harmonic surfaces, applying mouse transformations to coordinate fields, and computing functions of coordinate fields. Most of these auxiliary modules take AVS fields as input and produce fields as output. We prefer to write modules that operate on AVS fields and avoid writing modules that take geometry objects as input. This is because it is easier to process individual fields of a geometry specification (vertices, triangles, colors, etc.) rather than extracting this information from a geometry object and processing the selected data.

CHEMISTRY-SPECIFIC MODULES

AVS is a general-purpose visualization environment; it does not have elaborate data structures for specific applications areas. We extend this philosophy by developing general-purpose modules whenever possible. However, modules specific to biomolecular visualization must be added as a link between computational biochemists and the general-purpose modules.

We have developed several AVS modules for chemistry-specific tasks. Some of these modules were developed specifically as AVS modules, while others were stand-alone programs that were imported into the AVS environment. Examples of the latter include XFIT, an electron density fitting program developed by McRee,[22] and RIBBONS by Carson,[23] which were ported to AVS by Shah; and MSMS, a molecular surface program[24–26] written and ported to AVS by Sanner.

We have written other chemistry modules to combine and display normal modes,[17] and create images for texture mapping onto tubes and molecular surfaces.[16] Our primary chemistry modules are described in the following sections.

MCS TUBES module

The MCS TUBES module, written by Y. Chen, allows the selection of backbone or side-chain atoms that are displayed as tubes. It combines the functions of the POLYLINE, ELBOW, and POLYTUBE modules with a simple mechanism for selecting and coloring subsets of atoms.

MSMS module

Molecular surfaces are an important method of visualizing molecules.[27,28] Within AVS, we compute solvent accessible and solvent excluded molecular surfaces with the MSMS module written by M. Sanner.[24–26] MSMS takes as input atom coordinates and radii and computes AVS fields that define the surface vertices, normals, and triangle indices. The user can specify the radius of the solvent probe, the density of the surface triangulation, and which subset of atoms will be used to generate surface elements. MSMS labels each surface vertex with the index of the atom that generated it. Using this information, other modules can color the surface based on atom or residue properties. The

POLYGON module converts the surface description generated by MSMS into a geometry object that is rendered by the GEOMETRY VIEWER. Note that the input and output of MSMS are simple and do not directly refer to a molecular structure; the input is a set of spheres, and the output is fields of vertices and triangle indices. This design allows other general-purpose modules to operate on the MSMS input or output fields.

MV102 module

MV102 is a module for viewing molecular structures. It reads Brookhaven PDB files and displays molecules as lines or spheres. MV102 is the successor of the FFE program.[29] With MV102, users can control the color, geometric representation, and display mode of subsets of atoms. To specify these subsets, MV102 accepts a limited type of regular expression (atom expressions) similar to the regular expressions used in the UNIX shells to do pattern matching. Atom expressions contain fields for the molecule number, residue, and atom type. Each field is separated by a colon, and there are default values when using a partial atom expression. For example, the expression m1:ala1:ca selects the C_α atom of Ala-1 in molecule 1, and the expression m1::ca selects all C_α atoms of molecule 1. With atom expressions, users select subsets of atoms to be operated on by the MV102 commands. MV102 can control the display style (lines, spheres) and the color of the selected subset of atoms. MV102 allows the user to pick atoms, label atoms, and perform simple measurements of distance, bond angles, and torsion angles. MV102 also has the capability to control the display of a molecular surface computed by the MSMS module. MV102 can read molecule data from different files or from three input ports that specify the atom coordinates, bond connectivity, and atom names.

PLAY BINPOS module

The PLAY BINPOS module, written by J. Smith, displays trajectories of structures such as those computed by molecular dynamics simulations. PLAY BINPOS reads a PDB file that defines the structure under consideration and a binpos (binary position) file previously used by the FLEX program.[29] This module outputs the coordinates, bond information, and atom names for each instance of the trajectory.

An important feature is that the output trajectory can be used as input to the MV102 module. With this mechanism, we can filter and color instances of a trajectory using MV102 atom expressions. PLAY BINPOS runs as an AVS coroutine and the output can be filtered interactively. PLAY BINPOS has facilities for selecting individual frames and controlling the speed of the display.

PARTIAL SURFACE module

PARTIAL SURFACE (written by T. M.), uses atoms selected by MV102 and filters the output of MSMS so that only a subset of the entire surface is displayed.

VISUALIZATION EXAMPLES

AVS networks that perform different visualizations are described below. Elementary examples of our strategy, including examples showing rendering spheres, a backbone tube, bond connectivity colored by atom property, static coloring by sampling a three-dimensional scalar field, interactive coloring using a three-dimensional scalar field, visualization of time-dependent data using the ORTHOGONAL SLICER module, isocontours colored by gradient magnitude, and interactive control of surface triangulation, are presented in Ref. 1.

Surfaces colored by surface property

Figure 1 shows an AVS network to display an order 10 spherical harmonic representation of the crambin molecular surface (1crn.pdb) colored by the local surface shape.[30] The READ FIELD module at the upper right reads the surface coordinates and shape properties. The second READ FIELD module reads the *i, j, k* indices that define each triangle. EXTRACT SCALAR extracts the shape data. Colors are computed by COLORIZER. POLYGON produces a triangulated surface using the vertex, index, and color fields. Color Plate 1 shows the resulting image.

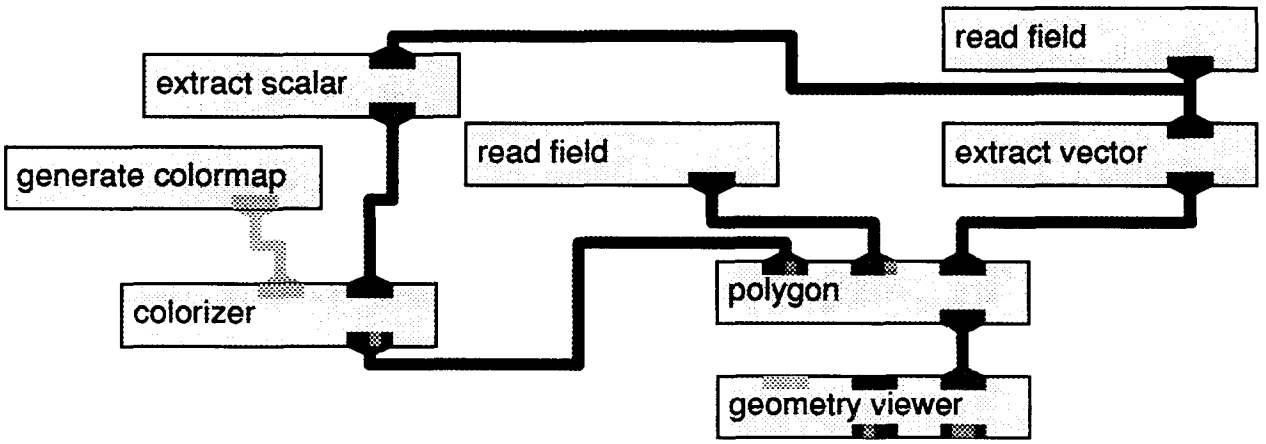

Figure 1. AVS network to color surfaces by a surface property.

Spheres colored by atom property

Figure 2 shows a network to display spheres of different radii that are colored by an atom property. In this example we show a united-atom CPK model of crambin color coded by the crystallographic temperature factor (*B* value). READ FIELD reads the coordinate, radius, and property data. EXTRACT VECTOR extracts the coordinates. The EXTRACT SCALAR module on the right extracts the atom radius. The EXTRACT SCALAR module on the left extracts the atom property. COLORIZER computes the colors. SPHERES creates colored spheres using the coordinate, radius, and color fields. Color Plate 2 shows the resulting image.

Intermolecular interfaces

One method to analyze the binding of proteins to other molecules is to compute a surface that represents the interface between them. Figure 3 shows a network to compute these interfaces. The INTERFACE module computes the locus of points midway between two interacting surfaces. This interface surface is clipped by a polytriangle clipping module to remove regions where the distance between the interacting molecules is greater than about 1.5 Å. The interface surface can be color coded by distance or by a physical property of one of the interacting molecules.

In this network, an interface is computed between two spherical harmonic surfaces. The two READ FIELD modules on the right read the vertex and triangle fields for one molecule and the two READ FIELD modules on the left read the corresponding data for the second molecule. The DOWNSIZEICO modules reduce the number of vertices and triangles of each surface and the POLYGON modules create geometry objects. The INTERFACE module takes as input two sets of the surface coordinates. In this example, the midpoint coordinate (rightmost output stream of the INTERFACE modules) is used to generate the interface geometry, and the distance between the surface points (leftmost output stream of the INTERFACE module) is converted from floating-point to integer and used as a vertex data field for the center POLYGON module. The CLIP module uses the vertex data information to clip triangles where the distance between the coordinate pairs is greater than some user-specified threshold, typically 1.5 Å. A simplified version of this network is shown in Figure 4, where the six modules that compute the interface have been combined into a single AVS macro module.

Color Plate 3 shows the resulting image. In this example, we show spherical harmonic surfaces for the X-ray crystal structures of bovine pancreatic trypsin inhibitor (BPTI; brown), trypsin, (magenta), and the intermolecular interface (blue). The crystal complex has been separated to give a clearer view of the interface region.

Molecular surfaces

Figure 5 shows a network to display molecular surfaces. The MV102 module reads a PDB file and assigns atomic radii. The MSMS module generates the molecular surface from the atom coordinate and radii data and computes fields that specify the surface vertices and triangles. POLYGON generates a geometry object from the surface description.

Color Plate 4 shows the molecular surface and covalent structure of crambin. For presentation purposes, the bonds were broadened using the standard AVS TUBE module and the surface was clipped using the standard AVS CLIP GEOM module.

Molecular dynamics

Figure 6 shows a network to display a set of molecular structures such as those computed from molecular dynamics simulations. The PLAY BINPOS module reads the trajectory information and generates an output field for the atom names, bonding information, and atomic coordinates. These fields are filtered using MV102 to select a subset of the entire molecule, control the atom colors, set the display style, and generate the geometry object.

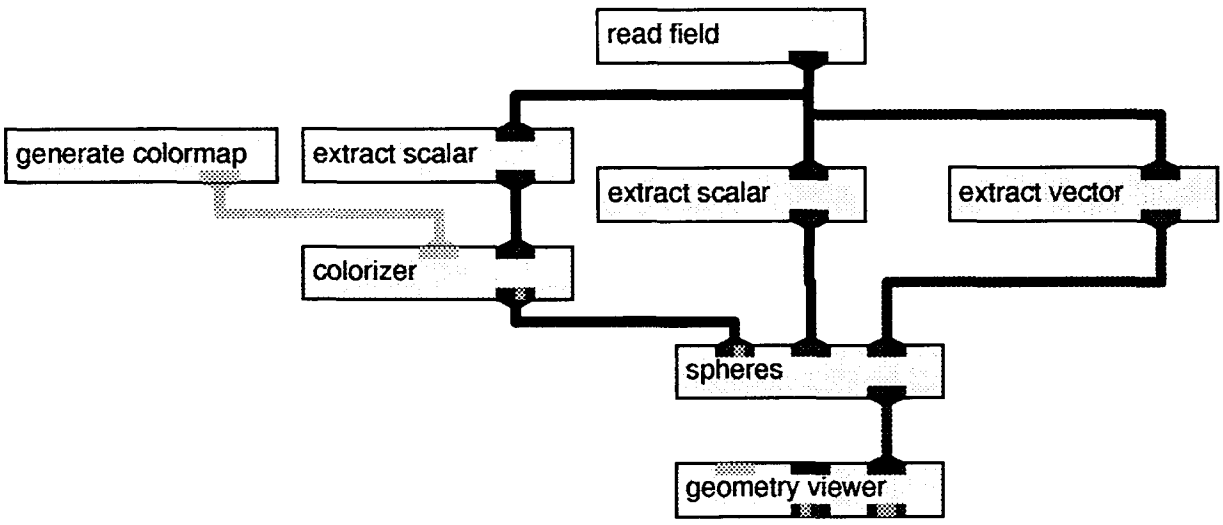

Figure 2. AVS network to color spheres by an atom property.

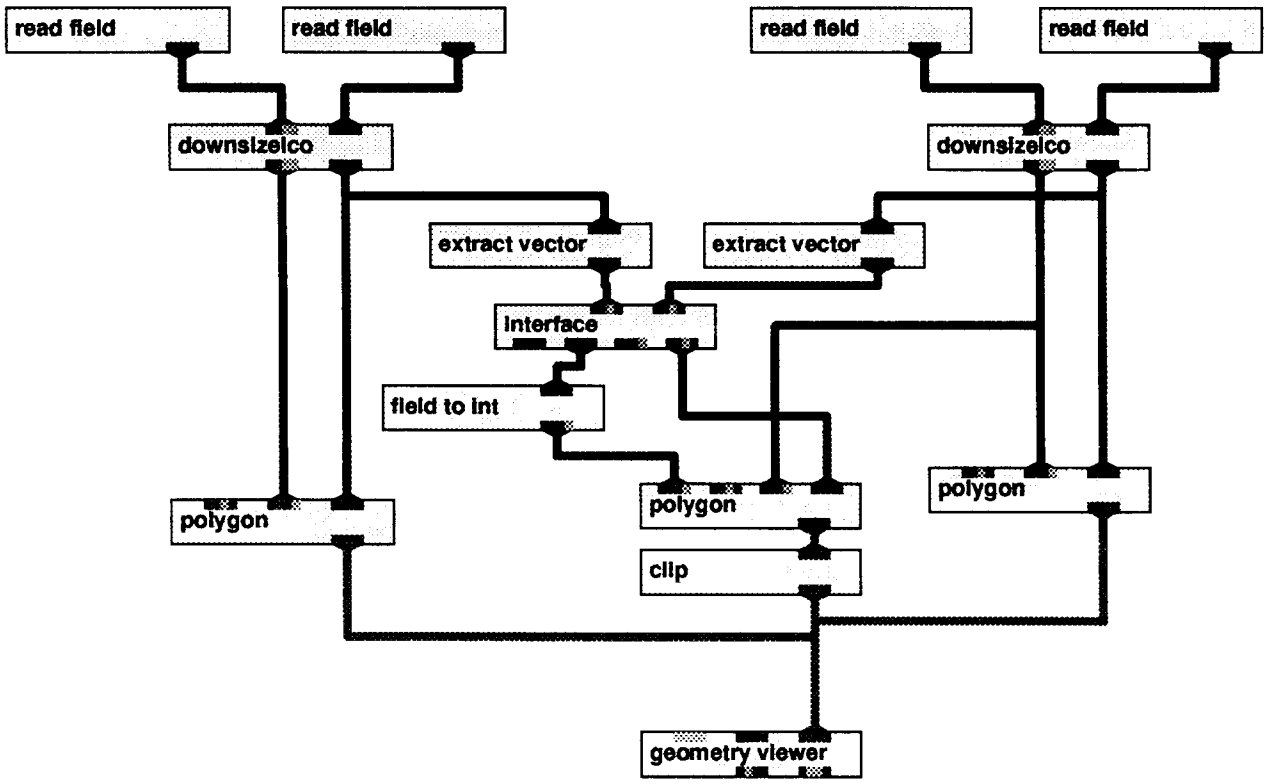

Figure 3. AVS network to display intermolecular interfaces.

Texture mapping

One novel visualization technique that is becoming increasingly popular in molecular graphics is texture mapping.[31,32] In the standard application of texture mapping, a two-dimensional image is applied to a three-dimensional surface. This is analogous to applying a decal to an object. We have developed techniques for defining surface texture coordinates independent of the surface properties; we call this location-based texture mapping.[16] Although the networks that implement these procedures are complex, AVS macro modules can streamline this procedure.

PROGRAMMATIC EXTENSIONS TO AVS

Most interaction with AVS is done using its graphical user interface (GUI). With the mouse, users select options, construct networks, transform objects, etc. AVS also provides a command line interpreter (CLI) where users type com-

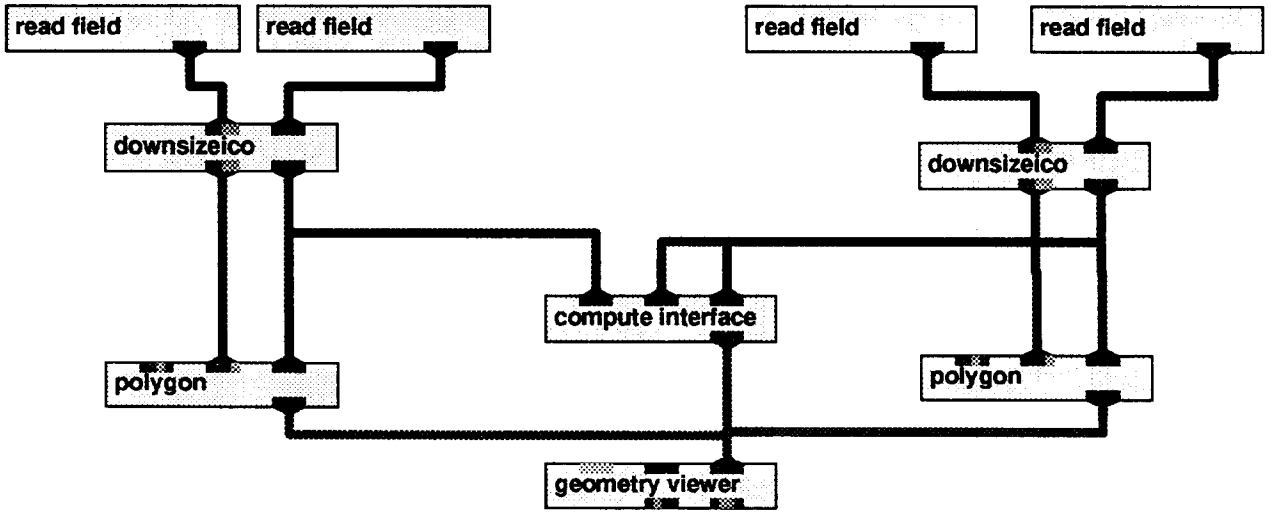

Figure 4. Simplified AVS network to display intermolecular interfaces.

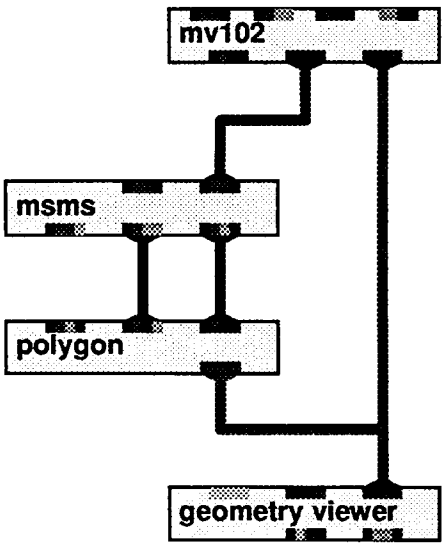

Figure 5. AVS network to compute molecular surfaces.

mands to AVS. Many, but not all, of the actions that can be done with the mouse can be done with the AVS CLI. Unfortunately, the AVS CLI is not a complete interpreter; it does not have facilities for expressions, variable assignment, loop control, or functions.

We have overcome some of these limitations using AVS-tool, a programmable environment that communicates with the AVS CLI over a TCP/IP connection.[33] AVS-tool is written in perl,[34] an interpreted language that combines some of the features of C, awk, sed, and csh. Using AVS-tool, we write perl functions to perform repetitive or complex tasks. An important feature of AVS-tool is that the user can load new perl functions into the current AVS-tool session. This functionality gives AVS functionality analogous to csh scripts in UNIX.

For example, one common task is generating a stereo image of the current scene. The core of the perl function that does this, with error checks and diagnostic output removed, is shown in Figure 7. `&IF('helix','off')`

We have developed other perl functions for tasks such as replacing failed modules in the current network, saving object, camera, or light transformations, and applying a function to a group of objects that match an input pattern. For example the AVS-tool command
applies the `off` function to all geometry objects whose name contains the substring `'helix`. These types of functions greatly simplify the use of complex AVS networks with many geometry objects.

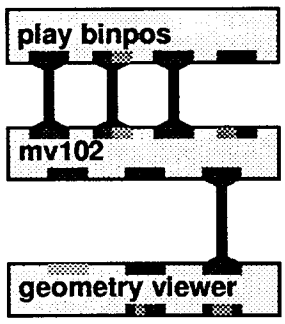

Figure 6. AVS network to display molecular dynamics trajectories.

We have also developed AVStk, a Tcl/Tk interface to AVS that implements the AVS-tool functionality using Tcl instead of perl.[35] The initial translation of AVS-tool to Tcl was done by J. Yng. An advantage of AVStk is that it is easy to write custom graphical user interfaces that can communicate with AVS and external applications. For example, we have developed a Tcl/Tk panel that combines many of the frequently used features of the geometry viewer panel. (This is useful because in AVS 5, the geometry viewer widgets cannot be controlled with the Layout Editor.) We have developed additional Tcl/Tk panels to control the basic operations of the stand-alone version of MSMS and the HARMONY suite of spherical harmonic programs.[12] We are currently formalizing the data communication protocol between our modeling applications, and believe that tool extension languages such as Tcl/Tk will be increasingly important for integrating diverse applications.

MODULE GENERATION LANGUAGES

The chemistry-specific modules described above were written using C, a general-purpose programming language.[36] For chemistry modules, we would prefer to use languages specifically designed for the applications area; C has no built-in constructs for chemistry.

The NAB (nucleic acid builder) language (designed and implemented by T.M.), is an alternate language for writing AVS modules.[37,38] NAB has a C-like syntax with extensions for operating on molecular data. NAB is implemented as a source-to-source translator that translates NAB code into C code. The resulting C program is compiled into an AVS module. NAB can also generate C code for stand-alone programs.

The advantage of this strategy is that the language used for creating the module, NAB, has data structures and operations designed specifically for molecular data. Its C-like syntax makes NAB constructs familiar to C programmers. NAB is analogous to high-level languages designed for other purposes. For example, awk[20] and perl[34] are high-level languages primarily designed to perform operations on ASCII files. An important difference, however, is that awk and perl are interpreted languages while NAB produces C code, which is then compiled.

An example of the use of NAB is presented below. To understand the relation between the structure of a DNA duplex and its four main geometric parameters (twist, *x* offset, inclination, and rise), we would like an AVS module that constructs duplexes and allows the user to modify each parameter interactively. Figure 8 shows the NAB program that generates this AVS module.

In this NAB program, the function `AVS_dna` takes as arguments the DNA sequence (represented as a character string) and the four parameters (represented as floating-point numbers). The `wc_complement` complement function generates the Watson–Crick complementary sequence using the input DNA sequence, and the `wc_helix` helix function constructs helical duplexes using the sequences of the individual strands and their geometric parameters.

The special comment lines, denoted by the keyword

```
# si.pl        save stereo images
#
# An AVS-tool perl function to save stereo pair to two files.
sub si {
    local($file,$directory) = @_;        # get 2 function arguments
    local($wi,$gv,$fullfile);            # define 3 local variables

    # get args and set defaults
    $directory = $si_directory unless $directory;   # set directory
    $directory = '/usr/tmp' unless $directory;      # set directory
    $file = 'test' unless $file;                    # set base name

    # add a "write image" module to network
    $wi = &module('add','write image');

    # find name of "geometry viewer" module in network
    $gv = &findmodule('geometry viewer');
    # connect "geometry viewer" output to "write image" input
    &cli("port_connect $gv:0 $wi:0");

    # right view (null rotation)
    $fullfile = $directory . '/' . $file . '.r.x' ; # generate filename
    &antialias('on');            # turn on global anti-aliasing
    &refresh;                    # render the scene
    &netwait;                    # wait for completion
    &param($wi,'Write Image Browser',$fullfile); # write right image to file
    &netwait;                    # wait for completion

    # left view (6 degree rotation)
    $fullfile = $directory . '/' . $file . '.l.x' ; # generate filename
    &rot('top',-6.0,'y');        # rotate scene
    &refresh;                    # render the scene
    &netwait;                    # wait for completion
    &param($wi,'Write Image Browser',$fullfile); # write left image to file
    &netwait;                    # wait for completion

    # cleanup
    &module('delete',$wi);       # delete "write image" module
    &antialias('off');           # turn off global anti-aliasing
    &rot('top',6,'y');           # rotate to original position
    &refresh;                    # render the scene
}
```

Figure 7. A basic AVS-tool perl function to generate stereo image pairs.

`AVSinfo`, specify information for creating an AVS widget for each of the four geometric parameters. The `parm` keyword specifies the parameter name and its initial, minimum, and maximum values. For the input sequence, a type-in widget is created automatically. Compilation of this NAB program yields the C source code for the AVS module named DNA.

Figure 9 shows how the resulting module is used in an AVS network. The DNA module reads the input sequence and the geometric parameters. Each parameter is connected to a dial widget that the user can adjust interactively. The module creates three output fields that contain the atomic coordinates, bonding information, and atom names. MV102 selects subsets of the DNA duplex, using its atom expressions, and converts the selected atoms into a geometry object for rendering. Color Plate 5 shows several DNA duplexes generated by the DNA module.

Our programmatic extensions of the basic AVS functionality take two forms: AVS-tool and AVStk control AVS using high-level interpreted languages (perl, Tcl), while NAB allows users to write modules using a high-level compiled language. An extension of this strategy is to develop a domain-specific interpreted language for the construction of AVS modules. Such a language would aid in the rapid

```
//    dna.nab
//
//    Creates an AVS data input module with a three output ports.
//    Has a "widget" for each parameter to AVS_dna.
//    The name of the module is "dna".

molecule    m;
molecule    wc_helix();
string      wc_complement();

//
// The following special comments set up the widget bounds:
//
//AVSinfo    parm    xoff    2.25 -5 10
//AVSinfo    parm    incl    -4.96 -20 30
//AVSinfo    parm    twist    36.0 20 45
//AVSinfo    parm    rise    3.38 2 4.5

molecule  AVS_dna( string seq,
    float xoff, float incl, float twist, float rise )
{
    string    cseq;

    cseq = wc_complement( seq, "bdna.std.rlb" );
    m = wc_helix(
            seq,  "bdna.std.rlb",
            cseq, "bdna.std.rlb",
            xoff, incl, twist, rise );
    return( m );
}
```

Figure 8. An AVS module specification using NAB.

prototyping of molecular computations and continue the trend toward high-level languages over general-purpose languages such as C.

SUMMARY

Our experience with AVS for molecular visualization has been generally positive, but AVS does have limitations for molecular work. Our networks often become large, making the visual tracing of the data flow difficult to ascertain.

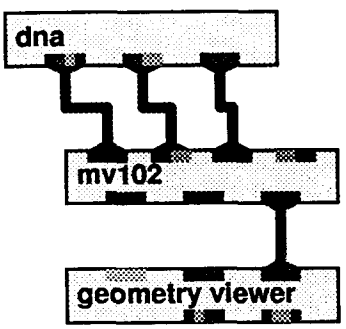

Figure 9. AVS network to create DNA duplexes using NAB.

Although AVS provides a macro module facility, the current version does not provide adequate interactive features to aid in the construction of macro modules. We believe that the software interface for constructing and modulating geometry data is awkward. We would like AVS to be more APL-like in its handling of scalars and multidimensional fields, and its compatibility between various Boolean, byte, integer, and floating-type primitive types.[39] To develop our software, we had to work around several AVS limitations and software bugs. AVS/Express overcomes some of these limitations.

The data structures of AVS do not permit operations such as inheritance or operations on classes of objects. The lack of tree data structures is particularly troublesome since this data structure is natural for molecular systems composed of atoms, fragments, residues, side chains, secondary structures, domains, subunits, and complexes. Methods to organize data in a hierarchical fashion would greatly enhance the applicability of dataflow architectures.

We have made AVS more suitable for biomolecular visualization by writing general-purpose and chemistry-specific modules, and by developing AVS-tool and NAB to facilitate network control and module development.

AVS is a powerful environment for data analysis and

visualization. Although AVS does not provide many facilities for manipulating molecular data, we have developed conventions and modules to analyze these types of data while retaining much of the flexibility and extensibility of the AVS strategy. AVS is particularly useful to view rapidly the results of molecular calculations and to aid in the development of other molecular software. We use commercial modeling packages when appropriate and AVS for tasks not easily performed by these packages. Although AVS has several limitations, the AVS environment is becoming an increasingly popular tool for the analysis and display of complex biomolecular data.

REFERENCES

1 Duncan, B.S., Pique, M., and Olson, A.J. AVS for molecular modeling. In: *Proc. AVS '93 Conf.*, Int. AVS Center, Research Triangle Park, North Carolina, 1993

2 Upson, C., Faulhaber, T.J., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., and van Dam, A. The Application Visualization System: A computational environment for scientific visualization. *IEEE Comput. Graphics Appl.* 1989, **9**(4), 30–42

3 Silicon Graphics Computer Systems. *IRIS Explorer 2.0 Technical Grade*. Silicon Graphics Computer Systems, 1992

4 Lucas, B., Abram, G.D., Collins, N.S., Epstein, D.A., Gresh, D.L., and McAuliffe, K.P. An architecture for a scientific visualization system. In: *Proceedings of Visualization '92*. IEEE Computer Society Press, 1992, p. 107

5 Rasure, J. and Young, M. An open environment for image processing software development. In: *Proceedings of 1992 SPIE/IS&T Symposium on Electronic Imaging*, Vol. 1659. 1992

6 Dyer, D.S. A dataflow toolkit for visualization. *IEEE Comput. Graphics Appl.* 1990, **10,** 60

7 Bowie, J.E. and Olson A.J. (eds.) *Data Visualization in Molecular Science*. Addison-Wesley, New York, 1995

8 Flurchick, K. and Bartolotti, L. Visualizing properties of atomic and molecular systems. *J. Mol. Graphics* 1995, **13**(1), 10–13

9 Walton, J. Visualization of sphere packs using a dataflow toolkit. *J. Mol. Graphics* 1994, **12**(4), 275–281

10 Goodsell, D.S. and Olson, A.J. Automated docking of substrates to proteins by simulated annealing. *Proteins* 1990, **8**(3), 195–202

11 Morris, G.M., Goodsell, D. and Olson, A.J. *AutoDock User Guide*. The Scripps Research Institute, La Jolla, California, 1995

12 Duncan, B.S. *Living in HARMONY: A Guide to Spherical Harmonic Representations of Molecular Surfaces*. The Scripps Research Institute, La Jolla, California, 1995 (in preparation)

13 Duncan, B.S. and Olson, A.J. Predicting protein–protein interactions using parametric surfaces. In: *Proceedings of the 13th Molecular Graphics Society Meeting*, 1994

14 Obeysekare, U.R. and Williams C.J. An application for visualizing molecular dynamics data developed under AVS/Express. In: *1995 AVS Users Group Conference*, 1995, Advanced Visual Systems, Waltham, Massachusetts, pp. 163–713

15 Obeysekare, U.R., Williams, C.J., and Rosenberg, R.O. Visualizing time dependent data from molecular dynamics simulations using AVS. In: *1994 AVS Users Group Conference*, 1995, Advanced Visual Systems, Waltham, Massachusetts, pp. 354–361

16 Duncan, B.S. and Olson, A.J. Texture mapping parametric molecular surfaces. *J. Mol. Graphics* 1995 **13,** 258–264

17 Duncan, B.S. and Olson, A.J. Approximation and visualization of large-scale motion of protein surfaces. *J. Mol. Graphics* 1995 **13,** 250–257

18 Olson, A.J. and Goodsell, D.S. Visualizing biological molecules. *Sci. Am.* 1992, **267**(5), 76–81

19 Olson, A.J. and Goodsell, D.S. Macromolecular graphics. *Curr. Opin. Struct. Biol.* 1992, **2,** 193–201

20 Aho, A.V., Kernigan, B.W., and Weinberger, P.J. *The AWK Programming Language*. Addison-Wesley, New York, 1988

21 Bernstein, F.C., Koetzle, T.F., Williams, G.J.B., Meyer, E.F.J., Brice, M.D., Rodgers, J.R., Kennard, O., Shimanouchi, T., and Tasumi, M. The Protein Data Bank: A computer-based archival file for macromolecular structures. *J. Mol. Biol.* 1977, **112,** 535–542

22 McRee, D.E. *Practical Protein Crystallography*. Academic Press, New York, 1993

23 Carson, M. Ribbon models of macromolecules. *J. Mol. Graphics* 1987, **5,** 103–106

24 Sanner, M.F. Modeling and Applications of Molecular Surfaces. Ph.D. Dissertation, Université de Haute-Alsace, France, 1992

25 Sanner, M.F., Olson, A.J., and Spehner, J.-C. Fast and robust computation of molecular surfaces. In: *Proc. 11th ACM Symp. Comp. Geom.*, 1995

26 Sanner, M.F., Olson, A.J., and Spehner, J.-C. Reduced surface: An efficient way to compute molecular surfaces. 1995 Biopolymers, In press

27 Richards, F.M. Areas, volumes, packing and protein structure. *Annu. Rev. Biophys. Bioeng.* 1977, **6,** 151–176

28 Connolly, M.L. Solvent-accessible surfaces of proteins and nucleic acids. *Science* 1983, **221,** 709–713

29 Pique, M.E., Macke, T.J., and Arvai, A.S. Flex: A light-weight molecular display program. *J. Mol. Graphics* 1991, **9,** 40–41

30 Duncan, B.S. and Olson, A.J. Approximation and characterization of molecular surfaces. *Biopolymers* 1993, **33,** 219–229

31 Catmull, E.E. A Subdivision Algorithm for Computer Display of Curved Surfaces. Ph.D. Dissertation, Univ. of Utah, Salt Lake City, Utah, 1974

32 Teschner, M., Henn, C., Vollhardt, H., Reiling, S., and Brinkmann, J. Texture mapping: A new tool for molecular graphics. *J. Mol. Graphics* 1994, **12,** 98–105

33 Duncan, B.S. and Olson, A.J. AVSTool: An interface

to the AVS CLI. In: *Proc. AVS '94 Conf.*, Int. AVS Center, Research Triangle Park, NC, 1994
34 Wall, L. and Schwartz, R.L. *Programming perl*. O'Reilly & Associates, Inc., Sebastopol, California, 1991
35 Ousterhout, J.K. *Tcl and the Tk Toolkit*. Addison-Wesley, New York, 1994
36 Kernighan, B.W. And Ritchie, D.M. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, New Jersey, 1978
37 Macke, T.J. *nab Language Reference*. The Scripps Research Institute, La Jolla, California, 1995
38 Macke, T.J. *nab User Manual*. The Scripps Research Institute, La Jolla, California, 1995
39 Iverson, K. *A Programming Language*. John Wiley & Sons, New York, 1962