WinMGM: A fast CPK molecular graphics program for analyzing molecular structure

M. Rahman and R. Brasseur

Laboratoire Chimie-Physique des Macromolécules aux Interfaces, Université Libre de Bruxelles, Belgium

A molecular modeling program is presented which has been written for Microsoft windows 3.1 and Windows NT operating systems. The program permits interactive molecular manipulation and also provides analytical tools such as energy computations and solvent accessible surfaces. An extremely fast algorithm is used which generates realistic space-filling CPK images in addition to wire frame, ribbons, MIDAS, labels, and points. An important feature of this algorithm is a highly optimized Z-buffer, which is described.

Keywords: CPK modeling, depth buffer algorithm, interactive application, molecular modeling

INTRODUCTION

Studies in molecular biology yield primary DNA and RNA sequences from which protein sequences are deduced. The increase in primary sequence data has resulted in a greater need for molecular modeling to study the conformation of peptides, drugs, and enzyme–substrate interactions. As a consequence, the demand is growing for molecular graphics workstations that allow the prediction and manipulation of these atomic structures.

Many modeling laboratories use high-speed graphic workstations, such as those of Sun, Evans and Sutherland, Silicon Graphics. These systems quickly draw wireframe representations, such that a thousand atoms can be drawn with real-time feedback. A disadvantage is that these systems are very expensive, which limits them to laboratories specialized in modeling. Moreover, they use a variety of incompatible operating systems (VMS or different versions of UNIX), which link the program to a specific computer.

The power of the desktop PCs has grown such that they can make computations as efficiently as a classic workstation. An i486 66-MHz-based PC has a power of 30 MIPS, which is the equivalent of a Sun IPX workstation. However, due to the absence of a graphic interface in the Microsoft DOS operating system, graphics are limited to a few specific modes, with limited resolution and colors.

Our purpose was to take the best of these two approaches: the low cost and versatility of PCs, and the power and conviviality of graphic interfaces which can support very high resolution or color graphic screens. To achieve this goal, we wrote a molecular manipulation and analysis program under Microsoft Windows 3.1 (using Win32s, which allows the execution of a 32-bit program in the 16-bit Windows 3.1 environment) and Windows NT operating systems. These two operating systems cover a very broad range of computers. Windows 3.1 allows the use of i386 with 4 Mo RAM PC, which is probably the most widely used computer in the world. Windows NT permits the use of more potent computers, such as the i486, Pentium, and RISC computers, the MIPS R4000 (the processor of silicon graphics), and the DEC alpha, which is the fastest existing processor. These operating systems also permit device-independent programs, that can yield on the standard VGA 640*480 16-colors mode up to 1600*1200 (or more) 16 million colors.

We also wanted this program to execute all the usual manipulations of molecular modeling: drawing one or more duplicate or different molecules, rotation, zoom, different drawing modes (wireframe, different types of balls, MIDAS, and CPK), and changing atom colors or drawing modes according to various criteria. In addition, analytic tools were also needed. This program cannot generate atomic structures, and requires entry files with atomic coordinates. For convenience, it reads standard PDB (Brookhaven Protein Data Bank) files, format-free text files that give atomic coordinates, and optionally, atomic symbols.

CPK DRAWING

In molecular graphics, two drawing modes are frequently used: wireframe and CPK space-filling models.

Color plates for this article are on page 198.

Received 11 January 1994; accepted 8 February 1994

Address reprint requests to Dr. Rahman at the Laboratoire Chimie-Physique des Macromolécules aux Interfaces CP206/2, Université Libre de Bruxelles, Brussels, Belgium.

Wireframe drawing provides fast pictures and transparency, which allows the study of the inner molecule. CPK (Corey–Pauling–Koltun) models are classical plastic models that chemists have used traditionally for the representation of molecules. These models consist of spheres representing atoms. They can be assembled to create a structure which is very similar to the true atomic structure. With the computer, we try to have the same quality of representation, giving the feeling that the computer screen is a picture of the model. The utility of such a three-dimensional representation makes this an advantageous feature.

We have implemented a high-quality CPK drawing feature that permits interactive manipulation, and requires drawing times of only a few seconds on a standard PC (i486 at 33 MHz).

Existing algorithms

Classical algorithms found in the literature present different methods to obtain CPK images, with various quality grades, computing times, and hardware requirements. The first method described was by Porter.[8] However, this algorithm is quite slow.

Implementations of CPK modeling were subsequently improved by use of the Z-buffer algorithm. The first was written by Shalloway[10] on an i286 PC, and was followed by the better known implementation of Palmer[7] on a Sun workstation. These methods, if well optimized, can produce CPK images in a few seconds.

Many other algorithms have been used to produce CPK images. Some of them, frequently slower, produce very high quality images. The main drawback of all these high quality results is that they are far too slow for an interactive implementation. The best known is the ray-tracing method (for a review, see Glassner[1]), which gives correct perspective, shades, reflections, and high quality images. Other methods give inferior quality images, but keep correct perspective and shading. One of them is implemented in the CONIC program, that computes analytical perspectives and shades.[5]

Algorithm used

We wanted to avoid three classical restrictions. First, we were looking for a system than can draw atom spheres in space-filling CPK, but that can easily be combined with other forms of representations (such as lines), yielding visible and hidden parts of a composite picture. Second, the system had to be fast enough to permit interactivity, if necessary, at the expense of lower image quality. Finally, it had to be device independent (in the number of colors and resolution).

In the text that follows, we make these assumptions: the X-axis is in the plane of the screen and extends to the right, the Y-axis is in the same plane and extends up, and the Z-axis is perpendicular to this plane and points toward the observer. The origin is in the middle of the screen.

We have written our CPK using a fast Z-buffer implementation. The principle of Z-buffer is to store 2 two-dimensional arrays with the same horizontal and vertical sizes as the drawing. One of these buffers is used to store the Z coordinate of the pixel, the other, its color. Initially, the color buffer is filled with the background color, and the Z-buffer with the lowest possible Z value. For each object to be drawn, the X, Y, and Z coordinates and colors of each point are computed. The Z coordinate of the point is compared with the corresponding Z value in the Z-buffer: if it is higher, it is visible; the Z in the Z-buffer is replaced by the Z of the point, and the same procedure is applied to the color. Otherwise, nothing happens.

When the Z-buffer is used, two assumptions are made: the projection is orthographic, so the radius of atoms does not vary with the Z coordinate, and the light source is at an infinite distance; thus the illumination does not vary from atom to atom. With these limitations, it is possible to precompute a template that contains, for a given atomic type, the Z and intensity of light for each pixel. This is possible because the sphere is perfectly symmetric, having shape and colors independent of its rotation. The Z-buffer algorithm is so applied to stamp these precomputed atoms on buffers, and once finished, colors are put back on the screen.

Because all computations for the drawing are fast integer comparisons between the Z of precalculated atom points and Z-buffer points, it can be very fast. Also, as the computation of the light intensity of an atom's points is not made during the drawing, sophisticated algorithms can be used to produce realistic colors and light reflections without extensive computing time. There are, however, limitations to this method: no shading appears and the perspective is not taken into account.

As we can see, the Z-buffer method, combined with the stamping method, satisfies perfectly our two first goals: they combine atom spheres in space-filling CPK with other forms of representations, and are fast enough to permit interactivity. But our third goal is not realized if a classical implementation is used. Indeed, the Z-buffer is frequently created to cover all the device surfaces, which is not very realistic for memory reasons if the image is gener-

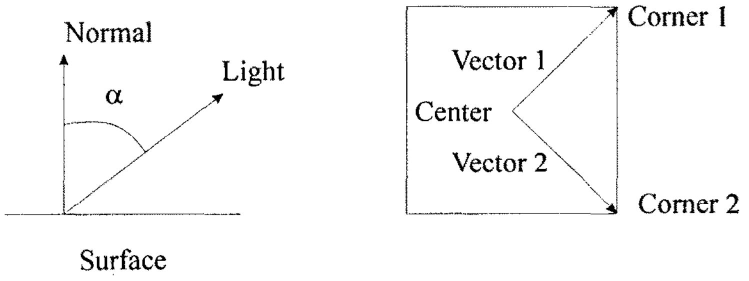


Figure 1. To compute the intensity of a square in the sphere, we compute the cosine of the angle α between the light direction and the normal (perpendicular) to the facet. If the light is parallel to the facet, α equals 0° or 180°: the cosine is zero. If it is perpendicular, α equals 90°, and the cosine is equal to 1. For intermediate angles, it lies between 0 and 1 (left). To compute the normal, two vectors (1 and 2) are defined, using the center of the facet and two of its corners. The vectorial product of these vectors gives the normal, which will be normed (its length set equal to 1). As the light source direction is already normed, the scalar product of these two vectors gives the cosine of the angles between them.

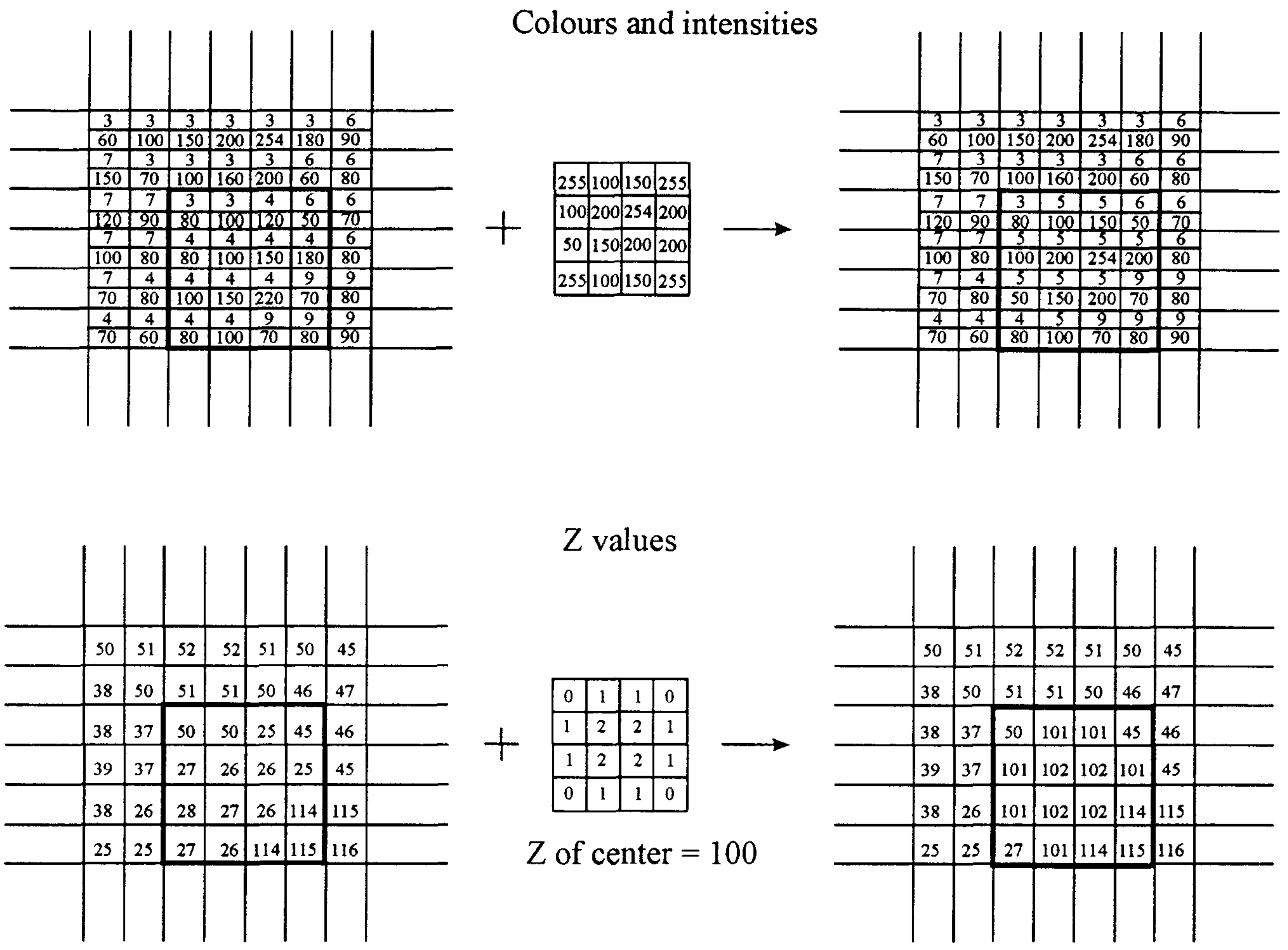


Figure 2. The stamping process is illustrated here. The original Z-buffer arrays are on the left, the precomputed atom that we are adding is in the middle, the result is on the right. For the new atom, we assume a color of 5, a radius of 4 pixels and the Z for its center of 100. The color values are on the top (color number in the upper part of the cell, intensity in the lower part); the Z values are on the bottom.

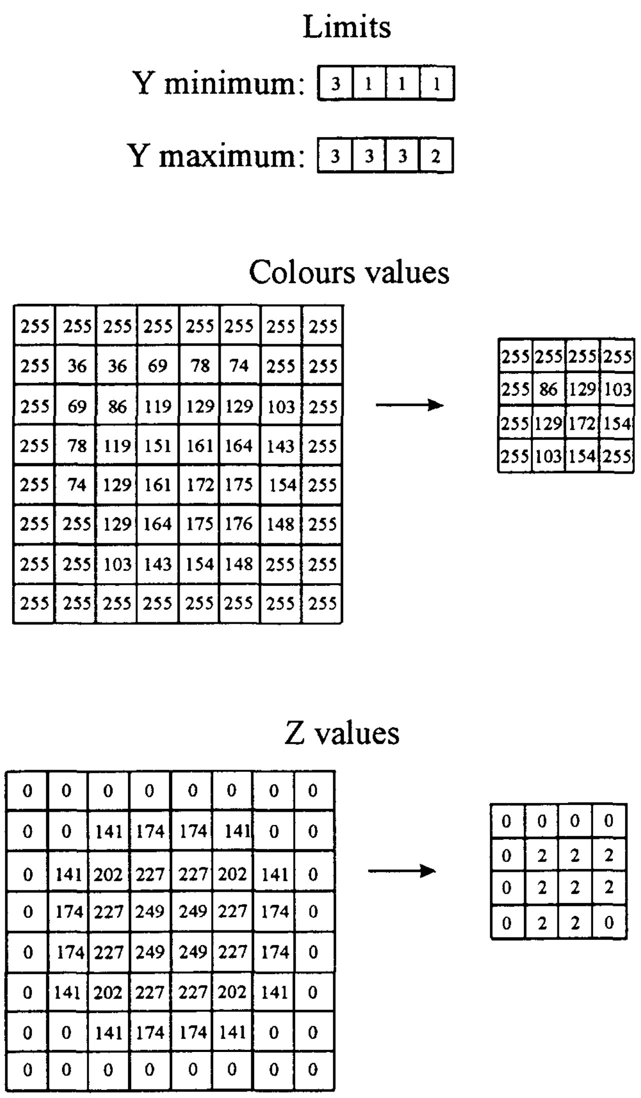
ated for high-resolution devices such as a laser printer. Therefore, we have introduced a technique already used in the Shalloway[10] implementation: we work with a moving Z-buffer, that covers only a part of the image, and once the Z-buffer is computed, we draw it. With this method, we can arbitrarily choose the memory needed for the arrays of the Z-buffer.

To use this algorithm, we must also have precomputed atom spheres at our disposition. As we don't know the resolution at which the image will be generated, we decided to compute the spheres at a resolution higher than any atom that will be drawn, thus avoiding a loss in quality. To achieve this, we have created two 256*256 tables, one with color intensities, the other with Z values. For memory reasons, these values are stored as unsigned bytes. The Z is in the range 0–255, and the colors in 0–254, or 255, in order to show that this pixel is not in the projection of the sphere.

The Z value is obtained by the classical equation of a sphere of radius 1 centered at 0: only the positive solution is kept, giving the point visible to the observer. With these equations, X, Y, Z coordinates for each point are obtained. If the point does not belong to the sphere (if the equation 1 has no solution), 255 is stored as color. The Z can directly be stored for the Z template, while the light intensity is computed by a formula based on the cosine of the angle between the normal to the point and the direction of the light source (Figure 1). Following the specular reflections desired, the color contrast, different balances can be applied. Here, we impose that all values (the cosine is in the range 0–1) that are lower than 0.975 are put in the range 0–0.7, and if higher than 0.975, are put in the range 0.7–1. This mathematical treatment allows us to lower the surface of a sphere that is very bright, and thus gives a better look to the atoms. Both computed tables are stored on disk in a binary form: they are loaded in memory at the initial step of the CPK drawing.

$$z = \sqrt[2]{1 - x^2 - y^2} \qquad (1)$$

To create the drawing, we have taken into account some characteristics of the Windows operating system. The drawing process can be summarized as follows: a band is prepared by the Z-buffer method using colors stored in the buffer. A device-independent bitmap (DIB) of band size is created with corresponding color values. This bitmap is drawn on the device using the appropriate Windows functions. The next band is treated, and so on. If the



Limits

Y minimum:	3	1	1	1
Y maximum:	3	3	3	2

Colours values

255	255	255	255	255	255	255	255
255	36	36	69	78	74	255	255
255	69	86	119	129	129	103	255
255	78	119	151	161	164	143	255
255	74	129	161	172	175	154	255
255	255	129	164	175	176	148	255
255	255	103	143	154	148	255	255
255	255	255	255	255	255	255	255

255	255	255	255
255	86	129	103
255	129	172	154
255	103	154	255

Z values

0	0	0	0	0	0	0	0
0	0	141	174	174	141	0	0
0	141	202	227	227	202	141	0
0	174	227	249	249	227	174	0
0	174	227	249	249	227	174	0
0	141	202	227	227	202	141	0
0	0	141	174	174	141	0	0
0	0	0	0	0	0	0	0

0	0	0	0
0	2	2	2
0	2	2	2
0	2	2	0

*Figure 3. The preparation of precomputed atoms of the right size. The original two-dimensional (256*256) arrays are on the left (atomic radius of 128); color intensities are on the top; and the Z values are on the bottom. Four arrays are used for each precomputed atom (real atomic radius): one for the intensities of pixel, one for the Z values (right arrays). Two one-dimensional arrays of the same width (top) are also used: one contains the minimum Y value that is part of the sphere; the other contains the maximum value (stored in base 0). In these tables, if the maximum index equals the minimum, no point is traced.*

device is a screen window, a particular process is followed: the function is used on a bitmap of the size of that window. The function initializes the bitmap in place of the window. When the drawing function is complete and the bitmap initialized, that bitmap is put on the window, which produces an instantaneous drawing in place of a series of bands.

We also tried to put the content of the band on the device pixel by pixel, but this approach (which is less memory consuming) was not further followed for performance reasons. The time necessary to generate the image became very important, due to the time needed to the execution of the SetPixel call (Table 1).

We have a precomputed atom, atomic coordinates, color codes, and radii of atoms to be drawn. The corresponding information for curves, points, and labels are also known. In the first step, the Z-buffer values are initialized to -32768. It is not necessary to initiate colors buffers, as we will see later. This buffer has a structure of two unsigned bytes: one that contains a color intensity (in the range 0–254) and one that contains a color number (0–15). This buffer is thus able to keep enough information to generate a 16.7 million color drawing.

For each atom, we first verify if there is an intersection with the current band, and compute the square that represents the intersection between the atom and the band. If there is no intersection, the next atom is analyzed. We also check that at least a part of the atom is between the back and front Z-clipping planes; if not, we continue to the next atom (Figure 2). At this stage, all the points of this square are analyzed. In our first implementation, the color from the current point was taken directly from the precomputed tables and tested to see if it was 255 (meaning that the point does not belong to the sphere). If the point is a part of the sphere, we proceed as follows: The Z-coordinate is taken from the precomputed table, corrected by the atomic radius, and added to the atom center. Through this method, we obtain the effective Z of the pixel. As we can do front and back Z-clipping, we verify that the point belongs to the correct range. If it belongs to the back, we continue to the next point; if it belongs to the front, we assign it a mean color intensity. The Z is compared to the Z-buffer value for this point. If it is in front, the color is put in the color buffer and the Z in the Z-buffer. The next point is then analyzed.

This implementation consumes relatively little memory, but is relatively slow. The reason is easy to understand. Accessing the good point in the precomputed tables needs a rule of 3 to convert from a radius of 128 to the atomic radius. The same procedure must be followed to convert the Z taken from the table to the Z of the atom. Moreover, the one-dimensional address in tables does not follow a regular progression from point to point, requiring one to compute each point from X and Y values. All these additional computations, including integer multiplications and divisions, drastically slow the program. Therefore, it is used only for very large atoms, whose radius is larger than 90 pixels. On a standard VGA screen, the width of a pixel is 0.28 mm: this cut-off value corresponds to atoms whose apparent radius is 0.28*90 = 2.5 cm.

Table 1. To show the speed evolution through the different versions of our program, we show here the time necessary to draw the crambin, a little protein of 324 atoms (PDB: 1crn). Times are given in seconds for a PC i486DX 33-MHz with 16-Mo RAM and a ISA Cirrus SVGA graphic card, used in the 800*600 65536 colors mode. Tests are done in the Windows 3.1 operating system, using Win32s. (1) First implementation, which uses directly the 256*256 precomputed tables and draws the Z-buffer by SetPixel calls. (2) Similar to (1), but the Z-buffer is drawn using a DIB. (3) Precomputed atoms of the exact radius are used when possible (small atoms, which is the case here). (4) Atoms which don't intersect a clipping plane are drawn without verification. (5) Final implementation. The atoms are sorted and drawn from back to front.

1	2	3	4	5
34.0	7.0	4.5	4.4	1.9

Table 2. Speed variations for different computers (all have a math coprocessor). Three molecules are used: aldomet is a drug of 28 atoms (atoms are too big for the fast mode), crambin is the same as in Table 1, CAPK is a big protein of 2822 atoms (PDB: 2CPK). The 386-33 is a PC i386DX 33-MHz with 8-Mo RAM and a ISA Trident SVGA graphic card; the 486-33 is a PC i486DX 33-MHz with 16-Mo RAM and a ISA Cirrus SVGA graphic card; the 486-66 is a PC i486DX 66-MHz with 16-Mo RAM and a VL-BUS Ati SVGA graphic card. All tests were performed in 800*600 256 colors mode, to allow comparison. Tests are done in the Windows 3.1 operating system and Win32s, and the Windows NT operating system. Times are given in seconds.

	386-33 Windows 3.1	486-33 Windows 3.1	486-33 Windows NT	486-66 Windows 3.1
Aldomet (46 pixels/A)	2.4	1.0	1.2	1.0
Crambin (18 pixels/A)	3.1	1.2	1.3	1.2
CAPK (6 pixels/A)	4.5	1.7	1.8	1.3

Table 3. Influence of the color number. We have drawn one molecule (crambin, the same as in Tables 1 and 2) with different numbers of colors, but keeping the same resolution. Tests were performed on a PC i486DX 33-MHz with 16-Mo RAM and a ISA Cirrus SVGA graphic card, using the 640*480 resolution, using the Windows 3.1 operating system and Win32s (actually, Windows NT does not support more than 256 colors with the Cirrus card, so we didn't use it). Times are given in seconds.

Colors	16	256	64 K	16 M
Times	3.1	0.8	1.2	1.2

Small atoms are precomputed at actual size before beginning the *Z*-buffer processing. This is carried out by taking the values from precomputed atoms previously created by a process similar to the stamping described above, except that all points are stored and that the *Z* of the atom center is not added. For time savings, we also store for each column, the *Y* minimum and maximum corresponding to the visible point (Figure 3). In the stamping process, only points that are between that range will be treated. As the surface of a circle is πr^2 and the square $4\pi r^2$, the fraction of points eliminated is $1-\pi/4$, i.e., 0.21. So, in the next operation, 21% of the points are not analyzed.

When these atoms are analyzed to create the *Z*-buffer, we verify if an atom and its corresponding radius are present. If they are not, the version of the algorithm described above is applied. Otherwise, we go from the left to the right edge of the square. For each *X* value, the *Y* limits from the precomputed atom are taken and we go from the bottom to the top visible *Y* (boundaries of the atom or of the square, following the atom position, radius, and band position). It is worth noting that two versions of these loops exist: one if a *Z*-clipping plane passes through the atom and one where the test for *Z*-clipping is removed.

As points in the precomputed atoms tables are continuous in the *X* and *Y* directions, the address computing can be simplified. The address corresponding to the current *X* and first *Y* in the *Z*-buffer and in precomputed tables is computed for each *X* value, before entering in the *Y* loop. Two addresses are set to their new values at the end of the *Y* loop, by adding the width of the corresponding buffer. This is faster than computing the address of a point not directly related to the previous one, as in the original implementation. The speed is also increased when the *Z* of the precomputed atom can, after addition of the *Z* of the atomic center, be compared with the *Z* in the *Z*-buffer, and stored if visible. For speed reasons, we also sorted atoms according to decreasing Z-value. With this sorting, the front atoms are drawn first, and atoms with lower *Z* values are drawn next, as their *Zs* are lower than the *Z* already stored in the Z-buffer. Most of their points (*Z* and color values) are not transferred, which speeds up the Z-buffer creation (Tables 2 and 3).

We have added to the three loops (for big atoms, small atoms with a clipping plane, and small atoms without a clipping plane) some color correction to take into account the *Z* of the atom and to give the front atoms more brightness than the back atoms. This correction is obtained by multiplying the precomputed intensity of the point by a factor ($\frac{5}{8}$, $\frac{3}{4}$, or 1) which depends on the *Z* of the center of the atom. As this multiplication is done by bit shifting, it is very quick, and gives the drawing a better visual depth.

As mentioned before, the Z-buffer creation can also deal with objects other than atom spheres. We have adapted our Z-buffer routine in such a manner that it can also treat lines (to make ribbons or to combine wireframe atoms with space-filling CPK), text (to display labels, such as amino acids or atom names), MIDAS (points to represent the Van der Waals surface of an atom), and points (to represent solvent accessible surfaces). All these objects are also treated for Z-clipping.

For line creation, the Windows function LineDDA is used which implements the DDA (differential digital analyzer) algorithm to find optimal points to link two extremities of a line. For each line segment to be drawn, that function is called, which gives us the coordinates of all intermediate points. The treatment applied to each of these points is identical to that applied to each atomic point in the implementation for big atoms. The *Z* of the point is computed by a linear interpolation between the *X* value of the point, and the beginning and end *X* and *Z* of the line. For text, we have followed an approach very similar to that of precomputed atoms, except that text is precomputed, creating a color buffer that indicates whether each point is shown or not, before each drawing. This is obtained by the recuperation, using Windows functions of the character font of the current device. The stamping is done as for small atoms without Z-clipping. One simplification occurs. As text possesses no depth, the *Z* put in the Z-buffer is the *Z* where the string must be placed. Drawing MIDAS and simple points is done in the same manner as for points given by LineDDA, except that in this case, three exact coordinates are known.

After creating the Z-buffer, a DIB bitmap must be created and drawn on the device surface. For the DIB creation, a distinction must occur between three cases: whether the device is palette based, and if it is not, whether it is monochrome. In a palette-based device, the color is given as an unsigned 1 byte number that identifies an entry in a color palette. The color palette contains the explicit color, specified as a RGB (red, green, blue) value. This palette system is mainly used by displays that permit 256 colors, such as most super-VGA modes. If the device is not palette based (16 colors, or true color displays with 32768, 65535, or 16.7 million colors), an explicit RGB value (3 bytes) is put in the DIB. If the device is monochrome (mainly printers), we put a byte value in the DIB, that represents a level of grey. In the three cases, the value is computed with color table constructed with the Z-buffer algorithm. For speed reasons, the DIB is initialized with the background color, and the color present in the color buffer is set only if the *Z* of the pixel is different from -32768, the initialization value. After this, the DIB is drawn using Windows functions.

Program features

As briefly mentioned in the introduction, our program comprises all basics of multimolecular manipulation: rotation, zoom, application of *Z* front and back clipping, displacement of one or more molecules using the mouse in real time. The picture is also selected by the user, on a molecule-by-molecule basis: atoms, ribbons, labels, atom names, and points. Ribbons are a generic system to trace colored lines, which may represent anything wanted. In our program, ribbons are used to represent the secondary structure of protein, molecular hydrophobicity, or electrostatic potential around a molecule. Points are used to represent the surface accessible to the solvent.

Various atoms representations can be selected, on a molecule-by-molecule, or atom-by-atom basis. The most commonly used modes are the wireframe and CPK modes. But other modes can be used, such as balls, which can be empty, filled, filled with a CPK look, and MIDAS, where the van der Waals surface of atoms is drawn as points. Ortep, which gives an impression of thickness to atomic bonds, can also be used in conjunction with ball modes. These modes can be combined as desired, except that ball modes are incompatible with CPK modes. For all modes, an orthographic or perspective projection is available, on a molecule-by-molecule basis.

The classic drawing consists in one representation of molecules in the program's window. For special purposes, however, special presentations have been written. For example, it is possible to create two different views, i.e., rotated from each other by 7° to give a stereoscopic view. Another capability is to draw the image in full screen, which facilitates making screen pictures. To make movies from views, automatic rotations around three axes can be obtained. The delay between two images and rotation angles around each axis can be determined by the user.

On a selection basis, all parameters from atoms, curves, and labels can be modified. This includes colors, visible or hidden, apparent radii for balls and the MIDAS mode, the drawing mode, bonds between atoms, coordinates, and atomic type. This system allows one to customize the representation of a molecule, but also to modify an existing molecule (by adding, removing, or modifying atoms). For drawing customization (colors and drawing modes), which are probably the most used features in this system (Color Plate 4), the selection has great importance. This selection can be done according to different criteria: position on screen (surrounding atoms with the mouse), PDB parameters if the molecule comes from the PDB (see below), atom number (one or a range), and type. Curves and labels can also be selected by number or range.

If the molecule comes from the PDB, additional manipulations can be carried out based on the information contained in the file. These manipulations consist of atom selection, deselection, coloring, and visibility. It is possible to act on atoms belonging to an amino acid sequence or a protein chain, atoms having a particular secondary structure (helix, sheet, or turn), atoms which are part of an active site, atoms that constitute the skeleton, or atoms that are part of a chemical function or a monomer (amino-acids, base, heme), or a family of amino-acids. With these functions, it is possible to place a molecular region of interest in the foreground, to study the configuration of an active site. The other group of functions based on the PDB are ribbon creation functions. Ribbons are smoothed lines or surfaces that represent the secondary structure of a protein, by following the path of α carbons. They are very interesting because they offer a view of a protein in which the folding is clearly shown, without any confusing atoms. Different types of ribbons can be created: a simple line, two lines a bit away from the path of α carbons, multiples lines which follow that path, and surfaces (Color Plate 1). These ribbons can be traced on all protein chains, on a particular chain, or on an amino acid sequence, and can be added to each other. They can be colored in a uniform manner, by amino acid or by secondary structure.

In order to study loaded molecules, three types of tools have been provided. These consist of the molecular electrostatic (MEP) or hydrophobic potential[2] (MHP), representing atoms in false colors to show their properties, and the solvent accessible surface. These tools give correct results only if the molecule is complete, i.e., if it contains all its atoms, and for energy or MEP calculations, its atomic charges. As PDB files frequently do not contain hydrogen atoms (X-ray diffraction does not give their positions, because they don't possess sufficient electronic density to lead to diffraction), and as other files do not necessarily contain them, we have added a function to add them to a molecule, respecting valence angles, bond lengths, and valence numbers. We have also added a function to compute approximate atomic charges, based on chemical functions.

MEP and MHP methods permit one to study the electric (MEP) and hydrophobic (MHP) environment around a molecule (peptide, drug, lipid, or protein). The MHP method has permitted classification of different types of peptide helices that interact with lipid membranes, and is frequently used to study peptide conformation, and interactions between peptides or peptides and lipids. In addition to methods described in original articles, we have added options to normalize the distance between the point where the potential is computed and the atom considered by the solvent diameter since the hydrophobic and electrostatic effects seem to decrease with the number of solvent molecules inserted rather than by the distance.[3] For the MHP method, it is also possible to normalize by the solvent accessible surface of atoms, because hidden parts (such as the amide skeleton of a peptide) do not seem to play a major role in interactions between molecules (Color Plate 2).[3]

The representation of atoms of a molecule in false colors permits the display of atomic properties. Colors can be chosen by atomic charge, hydrophobicity, accessible surface, energy level, total energy, or its components, hydrophobic, electrostatic, van der Waals, or solvatation energy (Color Plate 3).

The last tool we have added is the computation of the solvent accessible surface. Two surfaces can be computed. The first is based on the definition of Lee and Richard,[4,6] consisting of the trajectory of the center of the solvent when rolled over the molecule. The second is the surface defined by Richard,[9] which consists of points accessible to the surface of the solvent molecule. Both surfaces can be represented by dots. These are useful in studying interactions between molecules, such as between an enzyme and a substrate.

Conclusion

We have written a program that permits interactive manipulation of molecules in simple or sophisticated drawing modes incorporating a fast CPK algorithm. The program also includes basic analysis tools which allow the study of molecules coming from the PDB or from modeling. The program can be used as a front end to a computational program that computes molecular structures. Another use is in education, where it can be a low-cost tool to familiarize students with molecules and molecular graphics.

ACKNOWLEDGEMENTS

We are grateful to the Association Française de Lutte contre la Mucoviscidose and the Belgian Fond National de la Recherche Scientifique for financial support.

REFERENCES

1 *An Introduction to Ray Tracing.* (A.S. Glassner, Ed.) Academic Press, New York, 1989
2 Brasseur, R., Differentiation of lipid-associating helices by the use of three-dimensional molecular hydrophobicity potential calculations. *J. Biol. Chim.* 1991, **24,** 16120–16127
3 Brasseur, R., *Protein Engineering,* submitted
4 Finney, J.L., Volume occupation, environment and accessibility in proteins, environment and molecular area of RNase-S. *J. Mol. Biol.* 1978, **119,** 415–442
5 Huang, C.C., Pettersen, E.F., Klein, T.E., Ferrin, T.E., and Langridge, R., Conic: A fast renderer for space-filling molecules with shadows. *J. Mol. Graphics* 1991, **9,** 230–236
6 Lee, B. and Richard, F., The interpretation of protein structures: Estimation of static accessibility. *J. Mol. Biol.* 1971, **55,** 379–400
7 Palmer, T.C. and Hausheer, F.H., Context-free spheres: A new method for rapid CPK image generation. *J. Mol. Graphics* 1988, **6,** 149–154
8 Porter, T., Spherical shading. *Comp. Graph.* 1978, **12,** 282–285
9 Richard, F., Areas, volumes, packing and protein structure. *Ann. Rev. Biophys. Bioeng.* 1977, **6,** 151–176
10 Shalloway, D., Sneddon, S.F., and Little, E.K., Microcomputer-based three-dimensional stereoscopic macromolecular graphics display. *CABIOS* 1988, **4,** 193–196