Calculating CPK images on a UNIX workstation

Laurence H. Pearl

Cancer Research Campaign Biomolecular Structure Unit, Institute of Cancer Research, Sutton, Surrey SM2 5PX, UK

This article presents a simple, efficient algorithm for calculating shaded and depth-cued CPK pictures. The algorithm has been implemented in the C programming language on a UNIX workstation.

Keywords: hidden-point-removal problem, linked list, depth cuing

Received 7 December 1987
Accepted 6 January 1988

While molecular structures represented as vectors connecting atomic centers can be rotated and generally transformed in real time, they give a poor impression of the surface of the molecule. Dot surfaces[1-3] provide a reasonable improvement and are widely used in many interactive molecular modeling systems. But perhaps the most informative, and certainly the most attractive, representation of the surfaces of molecules is the computer graphics equivalent of the solid space filling models first used by Corey, Pauling and Koltun (hence CPK).

Raster graphics displays can greatly simplify the solution of the hidden-point-removal problem required when calculating the image of an opaque object. A very simple approximation to a CPK picture can be achieved by drawing a circle (or a set of concentric circles of differing intensity, to imitate a shaded sphere) for each atom in decreasing order of the distance of the atom from the plane of view (see, for example, the front cover of the *EMBO Journal*, volumes 4 and 5).[4] Thus, the pixels lit by the farthest atoms are simply overwritten by closer atoms that obscure them. While this method can generate pictures quite rapidly, it fails to take into account the intersection of the spheres used to represent the atoms, and its results (particularly for smaller molecules) do not stand up to close inspection.

To achieve a true CPK image, we must solve the hidden-point-removal problem for each point in the final image. Researchers have devised a variety of techniques to achieve this.[5-6] This paper describes a general and efficient computer graphics program for generating true CPK images on a UNIX-based raster graphics workstation.

ALGORITHM

Identifying surface pixels

To generate an effective CPK picture, we must solve the hidden-point-removal problem for an object consisting purely of spheres. To achieve this, we must determine which sphere has the largest (i.e., closest to the viewer in a right-handed coordinate frame) z coordinate, for the x and y coordinates corresponding to each pixel in the picture. We can achieve this by evaluating for each pixel:

$$q = r^2 - (s - X)^2 - (t - Y)^2$$

where r is the radius of the atomic sphere whose center has coordinates X, Y and Z, and s and t are the coordinates corresponding to the pixel. If $q \geqslant 0$, then the pixel will lie under the orthographic projection of that sphere with a z coordinate of Z + sqrt(q); otherwise, if $q < 0$, the atom does not map onto that pixel. Thus, the atom that produces the maximum value for Z + sqrt(q) determines the color and intensity of that pixel. However, even for a medium resolution picture with 512 x 512 pixels, this calculation must be performed for each atom in the molecule, for each of the 256K pixels!

Clearly, we can improve this situation dramatically by limiting the number of atoms that must be tested at each pixel and by identifying which pixels we must test in advance. To achieve this, we must first calculate the horizontal and vertical extents of each atom's projection, mapped onto the array of pixels forming the raster image. Then for each row of pixels in turn (arbitrarily from the top down), we need consider only those atoms whose vertical extents span that row. We'll place the atoms on a linked list (ATOM_LIST), sorted in decreasing order of their maximum vertical extent. Then for each row of pixels, we'll transfer atoms from the head of ATOM_LIST and add them to a second linked list (CURRENT_LIST) if their maximum vertical extent is above the current pixel row and their minimum vertical extent is below it. Once we encounter an atom on ATOM_LIST that is entirely below the current pixel row, it becomes the new head of ATOM_LIST, and CURRENT_LIST is complete. We'll sort atoms into CURRENT_LIST in increasing order of their leftmost horizontal extent. If the CURRENT_LIST is empty

(i.e., no atoms span that row), we'll set every pixel to the background color.

Having established a list of atoms that span the pixel row, we can perform an analogous process for each pixel in that row (arbitrarily from the left moving right). We'll copy atoms from the head of CURRENT_LIST to a third linked list (ACTIVE_LIST) if their minimum horizontal extent is to the left of the pixel and their maximum horizontal extent is to the right of the pixel. Once we encounter an atom on CURRENT_LIST that is entirely to the left of the current pixel, it becomes the new head of CURRENT_LIST and ACTIVE_LIST is complete. We'll sort atoms into ACTIVE_LIST in decreasing order of their precalculated maximum z coordinate. Then we'll evaluate the value q for each atom on ACTIVE_LIST, starting at the head of the list, until the maximum z of the next atom is less than the highest value of Z + sqrt(q) found for this pixel. The atom that produces the highest value for Z + sqrt(q) contributes this pixel. The evaluation of q rarely needs to be performed for more than two or three atoms per pixel. Once the pixel has been set, we'll update ACTIVE_LIST for the next pixel by removing any atoms that are now entirely to the right of the new pixel and add new atoms from CURRENT_LIST as above. Once a row has been processed, we'll reset ACTIVE_LIST to null, update CURRENT_LIST for the next row by removing those atoms whose minimum vertical extent is now entirely above the new pixel row and add new atoms from ATOM_LIST as described above. The algorithm terminates when ATOM_LIST is empty (see Figure 1).

Determining pixel intensity

Once we've identified the atom, whose surface contributes a particular pixel, we can determine the intensity of that pixel. The actual color for any given atom can be set according to the user's wishes, but it is the variations in intensity of those colors that give the appearance of shape and depth. A diffuse light source, coincident with the viewer, provides the clearest picture and is the most economical to calculate. The relative intensity of any surface point is then simply cos(Θ), the cosine of the angle between the surface normal to that point and the direction of illumination. You can introduce depth-cuing by decreasing the intensity of a pixel as a function of the z coordinate of the corresponding point on the surface. You can incorporate this into the angular shading by modifying the relative intensity from cos(Θ) to:

$$(1 - d + d^*(z - Z_{min})/(Z_{max} - Z_{min}))^*\cos(\Theta)$$

where Z_{min} and Z_{max} are the smallest and largest possible z coordinates for points on the surface of atomic spheres and d is a user-supplied contrast parameter varying between 0 and 1. A value of 0.75 to 1.0 produces the best effect for large molecules.

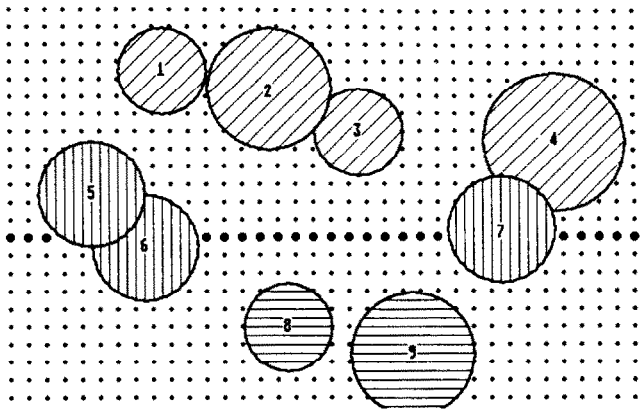

Figure 1. The circles represent the orthographic projection of the atom spheres onto the array of pixels forming the raster display. The bold dots indicate the row of pixels whose colors are being calculated. The atoms with diagonal shading are entirely above the active row and have been eliminated from the calculation. Atoms with vertical shading form the CURRENT_LIST, while those with horizontal shading form the ATOM_LIST, with atom number 8 at its head

IMPLEMENTATION

Programming

I wrote the program described above in C running under UNIX on a Silicon Graphics IRIS workstation. C is particularly suited to this type of application because it allows for structured data types of mixed composition and offers a flexible system of pointers.[7] I implemented the three-dimensional sorting of atom extents described above using linked lists of pointers to structures of the required data for each atom.

Specifying colors

The IRIS 2400 raster screen has a resolution of 1024 by 768 pixels and allows for the simultaneous display of up to 2^{12} different colors from a possible palette of 2^{24}. To simplify color specification without restricting the choice available, a separate graphics program is available that presents a projection of cubic RGB space down the white-black body diagonal onto a hexagonal lattice.[8] In this projection, a shaded sphere is drawn at each lattice point, within a triangle of 10 spheres along each edge. The color of each sphere is determined by its coordinates on three hexagonal axes (red, green and blue, respectively), such that the sum of the coordinates is 9 (Color Plate 1). Thus, pure red has coordinates 9,0,0, pure green is 0,9,0, pure blue is 0,0,9 and white has the coordinates 3,3,3. For each color, a maximum of 32 gray levels is provided, although an acceptable quality can be achieved with half that number.

Coloring atoms

You can specify the color of any atom in the picture using the alphanumeric data normally accompanying atomic coordinates in the Brookhaven Protein Databank

format.[9] Color specifications take the following form:

R,G,B,<SEQUENCE><RESIDUES><ATOMS>

where R,G,B is the color defined above. SEQUENCE is a comma-separated list of alphanumeric residue sequence identifiers, or a sequential range of residues given as A–B. You can use a wildcard symbol, *, to specify all residues or replace A or B in a sequence range to signify the first or last residue in a molecule. Chain identifiers (if present on the coordinate file) are added to the front of residue names, and whole separate chains can then be specified by use of the wildcard symbol such as A* or B*, and so on. RESIDUES is a comma-separated list of residue types, or * to signify all residue types. Similarly, ATOMS is a comma-separated list of atom names or * for all atom types. Atom names themselves can contain a trailing wildcard character that matches zero or more characters after the leading specified characters (e.g., C* matches C, C1 and CG1, while C1* matches C1 and C13, but not CD2). Atoms that are not matched by a color specification command are not drawn. Atoms can also be specifically omitted from the picture by a selection command where the R,G,B string is replaced with the exclusion symbol !. For example, you might use ! * HOH * to omit from the picture all water molecules found in a protein dataset. Trailing wildcards can be omitted from the selection command (e.g., 3,4,2 27–35 * * can be written as 3,4,2 27–35).

Color selection commands are applied in the order given, so that, for example, you can set a default color and then list specific selections or omissions. Commands are normally stored in a file that can be concatenated with other color files when the program is run (Color Plates 2–5).

Command lines and display preprocessors

In addition to the main program, several preprocessing utilities are available to perform a variety of transformations on the raw coordinate data. These transformations include generating a stereo pair, rotating about world axes and laying out several molecules on the screen. UNIX allows for a standard input and standard output to any program, which can be redirected to refer to files or connected by "pipes" (written as |) to the standard input and output of other programs.[10] For example, the CPK drawing program is normally invoked by the command

```
cpk -i<coordinates> -c<colorfile(s)> -t "title'
```

Two molecules can be drawn side by side by the command

```
layout <file1> <file2> -h2 -v1 |
cpk -c <colorfile(s)>
```

where layout is a preprocessor program that shifts the coordinates of the two input files so that they are side by side in the final picture. Layout can produce any rectangular arrangement according to the horizontal (-h) and vertical (-v) counts given in the command line (see, for example, Color Plate 6).

CONCLUSION

This paper describes a fast (< 30 seconds per picture on an IRIS 3000) and user-friendly program for generating high-quality CPK pictures with depth-cuing and a powerful color-selection system. The program is device-independent and can be ported readily to any computer with a C compiler and a high-resolution raster graphics screen and/or pixel-addressable hard-copy device.

ACKNOWLEDGEMENTS

I should like to thank Dr. Stephen Neidle for useful discussion and also thank the Cancer Research Campaign for its support.

REFERENCES

1 Langridge, R., *et al., Science*, 1981, **211**, 661–666
2 Pearl, L. H., and Honegger, A. *J. Mol. Graph.*, 1983, **1**, 9–12
3 Bash, P. A., *et al. Science*, 1983, **222**, 1325–1327
4 Islam, S., and Neidle, S. (unpublished)
5 Feldmann, R. J., Heller, S. R., and Bacon, C.R.T. *J. Chem. Doc.*, 1974, **12**, 234
6 Tickle, I. J., *et al. J. Mol. Graph.*, 1983, **1**, 68–70
7 Kernighan, B. W., and Ritchie, D. M. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1978
8 Foley, J. D., and van Dam, A. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, MA, USA, 1982
9 Bernstein, F. C., *et al. J. Mol. Biol*, 1977, **112**, 535–542
10 Kernighan, B.W., and Pike, R. *The UNIX Programming Environment*. Prentice-Hall, 1984