Fast algorithm for generating CPK images on graphics workstations

Chris Schafmeister

HyperCube Inc., Cambridge, Ontario, Canada

An algorithm for generating CPK images of molecules is presented that uses bitmap operations to solve the hidden-surface problem. The algorithm makes certain assumptions: that the size and shading of each atom are determined only by its type and are unaffected by the atom's position, and that an intersection table comprised of entries for all pairs of atoms whose radii intersect is available. For speed, bitmaps of shaded and colored atom disks are predrawn. The atoms are projected from back to front. Atom intersections are explicitly calculated and used to generate mask bitmaps that are then used to truncate the predrawn disks, leaving the visible portions of the atom disks.

Keywords: CPK modeling, raster graphics, interactive applications

INTRODUCTION

Of the various methods of representing molecular surfaces the one that is most familiar to chemists and, perhaps, the most pleasing to the eye is the CPK representation. Several researchers have developed methods for generating CPK images and each method places different burdens on computing hardware. Depth-buffer algorithms[1] require large amounts of memory, and for good performance require assembly language programming or special hardware. Scan-line depth-buffer algorithms[2] require far less memory than full depth-buffers but at the price of reduced performance. Area subdivision algorithms[3] are expensive in terms of computing power. The algorithm presented here uses a new approach. It calculates atom intersections explicitly and then uses standard bitmap operations to eliminate hidden parts of the atoms. These bitmap operations are available to programmers on most graphics workstations and personal computers, where they are either written in heavily optimized assembly language or implemented in hardware.

Address reprint requests to Chris Schafmeister at his present address: Department of Pharmaceutical Chemistry, University of California at San Francisco, San Francisco, CA, 94143-0446, USA.

Received 5 March 1990; accepted 5 July 1990

ALGORITHM

The algorithm is comprised of five steps: constructing the atom intersection table, predrawing bitmaps of shaded spheres, sorting the spheres by depth, calculating the intersections and drawing the shaded bitmaps on the display.

Atom intersection table

The algorithm calculates sphere intersections explicitly and therefore needs a list of atom pairs having overlapping radii. For computational efficiency each atom has a list of pointers to the other atoms that intersect it. There are two ways of obtaining the intersection table. Using van der Waals radii, an exhaustive search for intersecting atoms may be performed; however, the computation time is proportional to n^2 where n is the number of atoms, and for large numbers of atoms this process will dominate the overall computing time. Fortunately, this need be done only once for the entire molecule; only the entries for atoms that are moved (and those that they intersect) need to be updated subsequently. The other solution, which was used in the implementation described below, is to use radii that are approximately 75% of the van der Waals radii. By reducing the radii of the spheres it becomes more likely that sphere intersections will correspond to bonds between the atoms, and it becomes possible to approximate a full list of intersecting pairs with the bonding information. Using bonding information, however, makes it possible for artifacts to appear where two spheres intersect which have no bond between them. However in 'reasonable' molecules having unstrained bonds and bond angles this rarely happens. An example of a molecule that does have a slight artifact is cyclo-butane; the center of the image, where the four carbons overlap, will have an artifact in the form of a sharp corner.

Predrawn shaded bitmaps

Because large molecules usually consist of only a few different types of atoms, large speed increases can be realized by predrawing the colored bitmaps that will represent the different atoms. Predrawing the atoms forces certain assumptions to be made: The position of the atom does affect

neither its shading nor its size, which means that the shading model must assume an infinitely distant light source and that the model cannot be drawn with perspective. The shading model of Brickmann[4] was used in the implementation below, although any shading algorithm may be employed.

Depth sort

A depth sort is performed on the atoms to solve the hidden surface problem for nonintersecting atoms that obscure each other. The atoms will be projected back to front, with nearer atoms overlaying farther ones. The atoms are sorted on two keys: The primary key is the distance of the atom from the observer, the secondary key is the identity (ID) of the atom. In a right handed coordinate system where the x-axis is the horizontal and the y-axis is the vertical, the atoms should be sorted with increasing values of z (back to front) and in ascending values of atom ID. The atom ID is used to determine display order when two atom spheres have identical z values.

Sphere intersections

The hidden surface problem for intersecting spheres is solved by calculating the intersection and using it to mask the hidden portions of the closer atom's disk. Where two spheres intersect a circle is formed (Figure 1); when this circle is projected onto the viewing plane it forms an ellipse. A solution to the hidden surface problem is to first draw the disk of the farther sphere, then to clip the disk of the closer sphere with the front arc of the ellipse and finally to draw the clipped disk. However the solution of the intersection of an ellipse with a cricle involves a quartic equation and is too computationally expensive. For simplicity the portion of the ellipse that is used in clipping the disk is approximated by a circular arc.

The procedure for calculating the visible portion of an atom is as follows:

1 Create a blank (logical 0) bitmap for the front atom that has the same dimensions as the predrawn bitmap for this atom. Draw into this blank bitmap a filled white (logical 1) circle with the same radius as the filled circle in the predrawn bitmap. This bitmap will be called the "visible bitmap."

2 For every neighboring atom that intersects the front atom and precedes the front atom in the sort:

   a Calculate the large circle whose arc approximates the arc of the ellipse marking the boundary of the visible portion of the front atom's disk. (See appendix for derivation.)

   b Create a blank (logical 0) bitmap of the same dimensions as the visible bitmap and draw into it a filled white (logical 1) circle with the same radius as the calculated circle, offset from the center of the bitmap by the vector difference of the origin of the calculated circle and the origin of the front atom in the viewing plane. In most cases only a portion of the large filled circle will fall within the bitmap. This bitmap will be called the "mask bitmap." The radius of the calculated circle will approach infinity as the difference in Z values of the front atom's origin and the neighboring atom's origin approaches zero; in this event it is best to approximate the visible arc of the calculated circle by a straight line and to draw a filled quadrilateral within the mask bitmap rather than a filled circle.

   c Modify the visible bitmap by logically ANDing it with the mask bitmap. (See Figure 2.)

The visible bitmap is then ready to be used to draw the visible portion of the atom on the display.

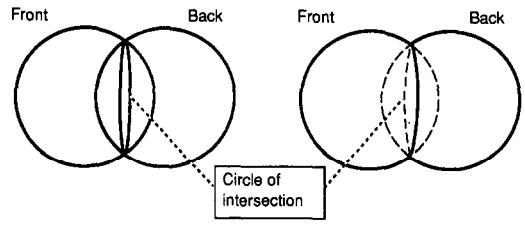


Figure 1. Illustration of how the circle of intersection formed where two spheres intersect projects onto the viewing plane as an ellipse. The solid lines in the figure on the right represent the outlines of the visible portions of the image. The visible portion of the ellipse can be approximated by a circular arc

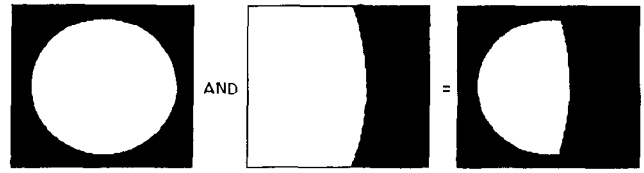


Figure 2. Logically ANDing the visible bitmap with the mask bitmap, which is generated from the calculated circle (the circular arc approximation to the visible portion of the sphere intersection), removes the portion of the front sphere that is buried within the back sphere

Drawing atoms

Once the visible bitmap has been produced for an atom, that atom is ready to be drawn on the display. The colored and shaded representation of the atom is retrieved from memory and is logically ANDed with the visible bitmap. The resulting bitmap is placed in a new bitmap (the "shaded bitmap"). The visible bitmap is then logically INVERTed and ANDed onto the display. The result of this is to create a black hole on the display of the correct shape (logical 0s) into which the shaded bitmap is logically ORed.

IMPLEMENTATION

The algorithm has been implemented under Microsoft Windows version 2.1, using Microsoft C version 5.1. It has been incorporated into HyperCube's HyperChem molecular modeling package. The implementation runs with less than 300 kilobytes of memory for bitmap storage. An implementation detail that had a great effect on the final per-

formance was the use of monochrome bitmaps for all bitmap operations except those involving colored and shaded bitmaps. On a display capable of 256 colors the color bitmaps have 8 bit planes; the use of monochrome bitmaps instead of color means that the bitmap operations will take one-eighth the time.

The color models were generated on a 20-MHz 80386 PC compatible with a math coprocessor. They were displayed on a Micrographics 16 color, 1280/1024 pixel display. Each took about 20 s to generate. On a 10-MHz 80386SX PS2 with a math coprocessor, 650 × 320 16-color graphics, a model of crambin containing 654 atoms, was rendered in 45 s at ~12 pixels/Å.

DISCUSSION

The difficulty of obtaining and implementing other CPK images rendering algorithms within MS Windows prompted the development of this algorithm and explains why there are no head-to-head timing comparisons. Table 1 lists operation unit count comparisons between the algorithm presented here and the Palmer–Hausheer (SphereBlt) algorithm[1] for the simple problem of two intersecting spheres. The comparison assumes that all of the predrawn bitmaps have been constructed. It also makes a few assumptions about the hardware. The first assumption is that the machine on which both algorithms are implemented has a 32-bit databus. This impacts the comparison in the following way: The algorithm presented in this paper uses bitmap operations that involve writing to, reading from and performing logical operations (AND, OR and INVERT) on monochrome bitmaps. A machine with a 32-bit databus is capable of performing these operations 32 bits at a time. The Palmer–Hausheer algorithm uses a SphereBlt operation that must deal with each pixel on an individual basis and should therefore take 32 times as long as a bitblt operation. Almost all workstations and high-end personal computers have 32-bit databuses.

The second assumption is that the display hardware is capable of 256 colors mapped into one byte per pixel. The algorithm presented in this paper treats color bitmaps as 8 monochrome bitmaps, requiring 8 times as much work. This assumption affects the implementation of the Palmer–Hausheer algorithm in that it requires only 1 write to the display memory for each pixel. Another common hardware configuration involves 256 colors mapped into 8 biplanes. This configuration would also require the SphereBlt implementation to perform 8 writes for each pixel. The

Table 1. Operation unit counts for a problem involving two intersecting spheres

Description†		Bitmap operations
Palmer–Hausheer[1] algorithm		
2 SphereBlts	Total	64
Explicit Intersection algorithm		
Draw the back sphere		
1 Draw the monochrome visible bitmap.		
Clear the bitmap, then draw the filled circle.	2	
2 Copy the predrawn bitmap into shaded.	8	
3 Logically AND shaded with visible.	8	
4 INVERT the visible bitmap.	1	
5 Logically AND display memory with visible.	8	
6 Logically OR display memory with shaded.	8	
Back sphere subtotal		35
Draw the front sphere.		
Drawing the front sphere requires the same amount of work as the back sphere (35 units) with the addition of three extra steps between Steps 1 and 2:	35	
a Calculate the intersection between the two spheres.		
Addition/subtraction	25	
Multiplication/division	41	
Square roots	4	
Transcendental functions	6	
Total		76 FLOPS
b Draw the monochrome "mask bitmap"		
Clear the bitmap, draw the filled arc.	2	
c Logically AND the mask bitmap with the visible bitmap.	1	
Front sphere subtotal		38
Total		71 + 76 FLOPS

†The Palmer–Hausheer implementation must manipulate each individual pixel, while bitmap operations can operate on 32 monochrome pixels at a time. Working on an identical size image, the SphereBlt operation will require the equivalent of 32 bitmap operations.

comparison in Table 1 uses the monochrome bitmap operation as the basic operation unit. This operation unit is very sensitive to implementation details and is not a constant unit, but can vary by up to a factor of three depending on the bitmap operation being performed. For the purpose of the comparison, the assumption is made that one SphereBlt operation requires approximately 32 units. For the simple problem described in Table 1, the Palmer–Hausheer implementation requires 64 units, while the implementation of the algorithm presented here requires 71 units and 76 floating point operations plus the initial overhead of the atom depth sort. In general, an implementation of the algorithm presented here requires the overhead of an initial depth sort of the atoms, 35 operation units for each sphere and 3 operation units plus 76 floating point operations for each intersection. The primary computational bottleneck is in generating and operating on bitmaps. The results of this operation count are combined with the fact that the Palmer–Hausheer algorithm requires a depth buffer with the same resolution as the screen, and a smaller depth buffer for each predrawn sphere. A 1024 × 1024 resolution 16-bit depth buffer requires 2 Mbytes of memory, and the predrawn spheres could easily require another megabyte; only a fraction of this is required for the predrawn colored bitmaps of the algorithm presented here.

This algorithm was developed with several requirements in mind: that it produce CPK images of reasonable quality; that it be easily implemented on personal computers and workstations having small memories and no three-dimensional graphics hardware; that it run at reasonable speeds

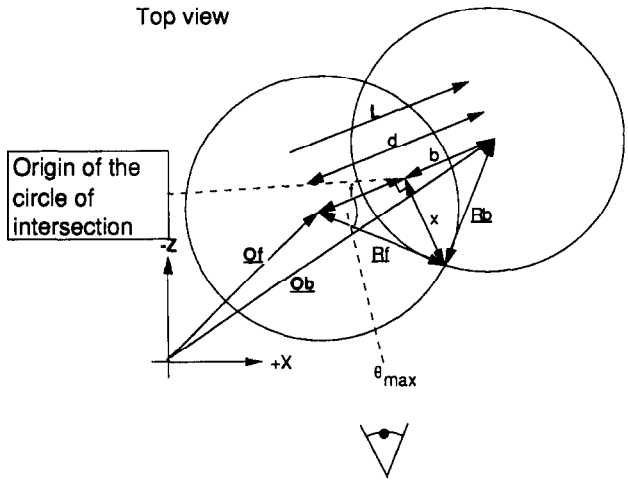


Figure 3. Top view of two intersecting spheres describing the known quantities (underlined) and calculated quantities (roman = scalar, bold = vectors). The vectors Of *and* Ob *are the front and back sphere origins;* Rf, *and* Rb *are the radii of the front and back spheres. From these known quantities the following variables are determined:* d, *the distance between the spheres;* **L**, *the vector difference between the sphere origins;* f *and* b, *the distances from the front and back spheres to the origin of the intersection circle, and* x, *the radius of the intersection circle. The variable* θ_{max} *is the angle between the line joining the origins of the spheres and the line between the origin of the front sphere and the perimeter of the intersection circle. (Figure 4 describes how it is used)*

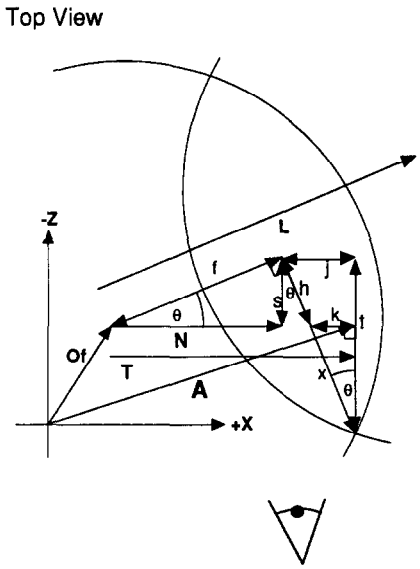


Figure 4. Magnified top view of two intersecting sphere: **N** *is the projection into the viewing plane of the vecto connecting the origin of the front sphere to the origin of th intersection circle. The variable* q *is the angle between and* **N**. *If* θ_{max} *is less than* θ *then the intersection will b hidden by the front sphere and need not be calculated. Th variable* s *is the* z*-component of the vector connecting th origin of the front sphere to the origin of the intersectio* j *is the distance from the projection of the origin of th intersection circle to the farthest point of the projection o the intersection circle from the origin of the front spher The variables* **T**, **A**, t *and* h *are then calculated, from whic one can calculate* g *and* k, *which are used in Figure 5 t calculate the radius of the intersection arc*

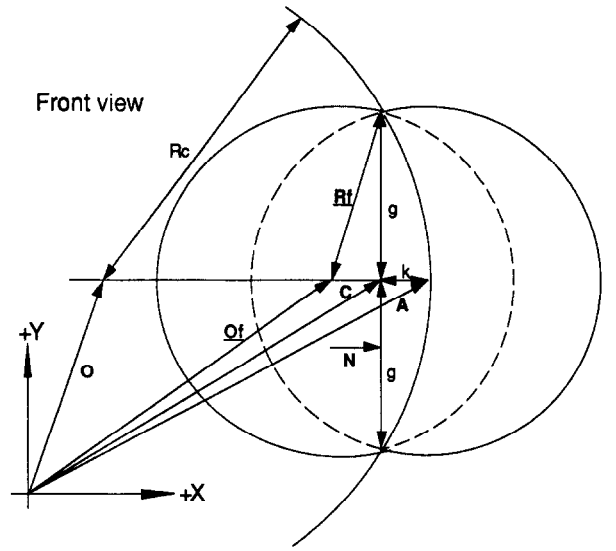
while being implemented in high-level languages; and tha it be device independent by leaving hardware details to th graphical user interface. Although the trend in computin is toward larger memories and more capable graphics hard ware the vast majority of personal computers and worksta tions currently have either no or only two-dimensional graphic hardware. In addition, the current trend toward the use o graphical user interfaces that typically have efficient bitma operation library routines, makes this algorithm an attractiv alternative on smaller computers.

The images produced by this algorithm have aliasing ar tifacts that are inherent to the algorithm. One possible so lution, which has not been extensively explored, is to us the visible arc of each calculated sphere intersection as path along which pixel color and shading can be adjuste to remove some of the aliasing artifacts.

APPENDIX

The following is pseudocode of the algorithm for calculatin intersections; please refer to Figures 3–5 for the meaning of the symbols. Underlined symbols represent quantities tha are known before the algorithm begins, bold symbols ar vectors, and normal symbols are scalars. From the origin and radii of the two intersecting spheres the algorithm ger erates either the origin and radius of the large calculate

Figure 5. Front view of the two intersecting spheres; the left sphere lies in front of the right sphere. The parameters g *and* k *describe the extents of the visible portion of the intersection circle in the viewing plane. The variable* k *is the maximum perpendicular distance from the chord to the arc, and* g *is 1/2 the length of the chord. The variables* g *and* k *are used to calculate* Rc, *the radius of the calculated circle. The origin of the calculated circle is calculated from* **A**, **C** *and* Rc. *In the case where* Rc *becomes very large, the calculated circle can be approximated by a quadrilateral with an edge on* **C** *perpendicular to* **N**, *and the quadrilateral filled with logical 1s on the side of the edge pointed away from by* **N**



circle whose arc is used to clip the visible bitmap or the line which marks the boundary of the quadrilateral which clips the visible bitmap if the calculated circle becomes too large.

```
L      = Ob − Of
d      = |L|
if (Rf + Rb ⇐ d) goto DONE
θ      = ArcSin(−Lz/d)
f      = (Rf² + d² − Rb²)/(2*d)

x      = (Rf² − f²)^1/2
θmax   = ArcSin(x/Rf)
if ((f > 0) AND (θmax ≤ θ)){
    goto DONE
}
N      = L * (f/d)
s      = ABS(Nz)
Nz = 0
j      = x * Sin(θ)
if (f < 0) then {

    k = j
    g = x
    j = −j
}
T = N * (1 + j/|N|)
A      = Of + T
if (f < 0) goto DRAW
t      = x * Cos(θ)
k      = (t − s) * Tan(θ)
h      = s/Cos(θ)
g      = (x² − h²)^1/2
DRAW
    M = −N/|N| * SIGN(f)
    C = Of + N
    if (2 * k < VERYSMALL) {

        Rc = MAXRADIUS + 1
    } else {
        Rc = (g² + k²)/(2 * k)
    }
    if (Rc ⇐ MAXRADIUS) {
        O = A + M * Rc
        **Draw circle with radius Rc at O
```

**Make sure the spheres are intersecting
**Lz is the z-component of the L vector
**f is found by solving the set of equations:
**$f^2 + x^2 = Rf^2$; $b^2 + x^2 = Rb^2$; $f + b = d$
**f can be less than zero if $Rf^2 + d^2 < Rb^2$,
**which occurs when small atoms are bonded to
**large ones; eg, Hbr

**Stop if the intersection is completely hidden

**Nz is the z-component of N
**Now N corresponds to the diagram

**In the case of a small sphere lying in front of
**a large sphere special processing must be done

**$s = h*\cos(\theta)$

**If the circle becomes too large then
**it is approximated by a quadrilateral

```
    } else {
        **Draw a quadrilateral with one edge on C,
        **perpendicular to N and shade the
        **side of the edge that N points away from
    }
DONE:
```


ACKNOWLEDGEMENTS

I would like to thank Graham Hurst for his constant encouragement and Neil Ostlund for his support. I would also like to thank the reviewers of this paper for their suggestions. This work was supported by the Industrial Research Assistance Programs (IRAP) of the National Research Council of Canada.

REFERENCES

1 Palmer, T.C. and Hausheer, F.H. Context-free spheres: A new method for rapid CPK image generation. *J. Mol. Graph.* 1988, **6**, 149

2 Porter, T. Spherical Shading. *Comp. Graph.* 1978, **12**(3) 282

3 Max, N.L. ATOMLLL: Atoms with Shading and Highlights. *Comp. Graph.* 1979, **13** 165

4 Brickmann, J. *J. Mol. Graph.* 1983, **1**, (3) 62