Fast algorithm for exact rendering of space-filling molecular models with shadows

Zhuming Ai and Yu Wei

Lab of Molecular and Biomolecular Electronics, Southeast University, Nanjing, P. R. China

An algorithm for accurate rendering of space-filling molecular models with shadows is presented. The intensity of light and cast shadows are computed to generate realistic pictures. Arbitrary numbers of light sources, which may be at infinite or finite distances can be applied. Hidden-surface removal, lighting, and shadowing are presented in detail.

Keywords: space-filling molecular model, shadow, sphere

INTRODUCTION

Space-filling molecular models are the simplest as well as the most informative representations of molecular structure, conformations, volumes, and surfaces. However, the models displayed on computer screens seem rather flat and do not convey depth relationships efficiently. Three methods are currently used to explore the three-dimensional (3D) information of the models: stereo pictures, animated pictures, and realistic rendered pictures. The former two methods make use of the brain's capacity to combine multiple two-dimensional (2D) pictures into one 3D image, and usually dedicated computer graphics hardware is needed to achieve stereo representation or animation. Realistic rendering, here, means the generated image is similar in appearance to a photograph of a plastic model. In realistic representation, cast shadows are of great importance because of their value for giving the depth perception cues.

Many methods have been used to generate space-filling models. One of the first algorithms for generating realistic images was by Porter.[1] The algorithm required that the observer and a single light source be coincident and infinitely far away from the molecule. So there are no shadows in the image.

Huijsmans has proposed a simple method for generating space-filling models which is suitable for microcomputers.[2] The algorithm cuts the 3D spheres into slices, and draws them from back to front, each of the slices partly overwriting the interior of the circular pattern displayed so far. It has the same limitations for observer and light source positions as Porter's algorithm, and shadows are not possible.

Huang has described the CONIC program, which renders shadow-cast and multilight source representations of space-filling molecular models.[3] The algorithm is scan line-based and closely parallels Porter's algorithm except that the constraint of observer and light source were removed. The computation time is more dependent on the fraction of the image that the atoms cover than on the number of atoms. Thus the computation time will not be reduced much for small molecules.

Other algorithms using ray tracing,[4] or a context-free spheres algorithm[5] have also been published, and all the implementations have common drawbacks: They are time consuming and they generally require extensive anti-aliasing.

The algorithm we present in this paper is a rapid computing method for rendering exact images of space-filling molecular models in which the number and position of the light sources can be chosen and located at finite or infinite distance. The algorithm contains three main aspects, hidden-surface removal, lighting, and shadowing. They have been combined in one efficient procedure. The algorithm avoids all the assumptions which other methods have used for hidden surface removal and shadowing, where these may cause a decrease of accuracy.

In the following sections, each aspect of the algorithm will be discussed and then the general algorithm will be presented.

HIDDEN-SURFACE REMOVAL

When creating spheres and intersecting spheres in space-filling molecular models, one has to remove the parts of the sphere surfaces hidden by other spheres which should not be visible from the viewpoint. One of the many different methods used to achieve this aim is Huijsmans' temporal priority algorithm; this is a shaded as well as intersected sphere hidden-surface algorithm. The algorithm is fast enough to create realistic space-filling molecular models on a personal computer. Huijsmans' algorithm cuts a 3D sphere into slices. The projected shading pattern on the circumference

Color Plates for this article are on page 190.

Address reprint requests to Dr. Ai at the Lab of Molecular and Biomolecular Electronics, Southeast University, Nanjing 210018, P. R. China.

Received 29 September 1992; revised 20 November 1992; accepted 24 November 1992

of the slices can be represented by an ordered set of infinitely thin filled circles each with their own depth coordinate. With this concept of slicing spheres and replacing them with a set of filled circles, nearby atoms may be looked upon as sets of slices that are interleaved. As the compiling of these interleaved sets of filled circles proceeds, an approximation of the intersection boundaries is automatically obtained as the result of overwriting.

However, enough slices must be cut for exact rendering of the resulting display along the edges of intersection. One sphere with radius of R, when transformed into screen coordinate system, must be cut into R slices represented by R circles with their corresponding radii ranging from 1 to R. So for a molecule with N atoms, the total number of circles will be

$$M = \sum_{i=1}^{N} R_i \tag{1}$$

This requirement causes two problems. First the memory requirement is very large. Second, and perhaps the more serious one, the computing time for sorting these slices is very long!

We use a dynamic cutting method to solve the problems (Figure 1). Because the drawing order of the slices cut from a sphere is fixed, we only need to cut and record one slice at a time for each sphere; this is the next slice to be drawn. After it has been drawn, we can cut the next one, until the sphere has been cut into R slices. So only N circles need to be recorded in the algorithm. The N circles should be sorted according to their depth, and the sorted list should be updated after each cutting. Because only one element in the list is out of order, the sorting process is very fast.

In principle, the dynamic cutting can proceed either from back to front or from front to back. If the cutting proceeds from back to front, each circle representing the slice partly overwrites the interior of the circular pattern displayed so far. Then if the light source is not coincident with the viewer, pixels on the same circle will have different colors, and the color value must be calculated even if this pixel will be overwritten later. In contrast if the dynamic cutting proceeds from front to back, we need only check whether each pixel on the circle has already been drawn. Only those pixels which have not been drawn need to be plotted and their color value calculated. So the superiority lies in the comparison of the time needed to calculate the pixel value and to check if the pixel has been drawn. If there is only one light source coincident with the observer, the back-to-front procedure is better because the color value is the same for all the pixels on a circle. However, for creating a model with shadows, there will be light sources not coincident with the observer, and the front-to-back policy is superior.

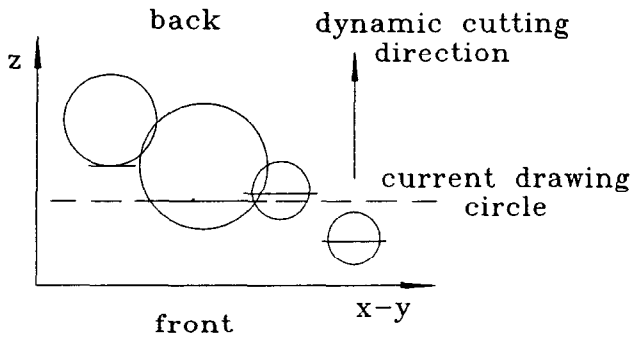

Figure 1. Only one circle need be recorded for each sphere, and the drawing order is dynamically updated. Note that only front half-spheres need to be sliced, because back half-spheres are not visible.

Compared with Huijsmans' algorithm, the algorithm proposed here greatly reduces the memory required, from the factor M to N. In systems with graphics processing units, the algorithm can use the central processing unit and graphics processing unit in parallel because of the parallelism in the sorting and pixel drawing.

LIGHTING AND SHADOWING

In Huijsmans' algorithm, all the pixels on a circle have the same color value. The color value is assigned to accomplish the Lambert's cosine law about the reflection of the light, with a single light source coincident with the observer and infinitely far away from the molecules. It takes into account the kind of atom: different kinds of atoms can have different parameters of red, green, and blue portion of light, and thus have different colors. There are no shadows in the image.

Lighting

In our algorithm, the color value of each pixel can be calculated separately according to the light. Arbitrary number of light sources can be implemented and the light sources can be at finite distance. The equation used for each red, green, and blue portion is[6]

$$I = k_a I_a + \Sigma k_d (\mathbf{L} \cdot \mathbf{N}) \tag{2}$$

where $k_a I_a$ is a constant representing the ambient reflection portion, and the summation represents the diffuse reflection portion. The parameters k_a and k_d are properties of the surfaces and may vary from sphere to sphere, or they may be set according to the kind of atom or user interest. Each item in the summation represents the diffuse reflection of a light to the pixel if it is not in the shadow of that light, otherwise the item will be zero. The detection of shadow will be discussed later. The vector **N** in the summation is normal to the sphere surface at the pixel position,

$$\mathbf{N} = \text{coordinate}_{\text{pixel}} - \text{coordinate}_{\text{center of sphere}} \tag{3}$$

The vector **L** is the direction of light source relative to the pixel

$$\mathbf{L} = \text{coordinate}_{\text{light source}} - \text{coordinate}_{\text{pixel}} \tag{4}$$

If the light source is at infinite distance, **L** will be constant.

Of course the other portions such as the specular reflection portion in the light model can also be included in Equation (2) without any difficulty; we omit them just to save computing time.

Shadowing

The usual method for shadowing is to apply hidden-surface removal in the direction of the lights, but it is time consuming. In the space-filling model all the objects in the

scene are spheres, so a more efficient method can be used.

For each point on the sphere, if it is in the shadow of a light source, it may be either in the shadow of itself or in the shadow of other spheres.

The first situation can be detected by checking the angle between the sphere surface normal at this point and the direction of light source (Figure 2). If the angle α is greater than 90° or

$$\cos\alpha < 0 \tag{5}$$

this point is in the shadow of its sphere.

In the second situation, the light will intersect with other spheres between this point and the light source. This can be detected by two steps: First, if a sphere is farther from the light source than the point, the point will not be in the shadow of the sphere. This can be detected by checking the angle between the direction of light source and the line connecting this point and the center of that sphere (Figure 2). If the angle β is greater than 90°, or

$$\cos\beta > 0 \tag{6}$$

the point is not in the shadow of that sphere. Second, in the remaining spheres, if the distance between the center of the sphere and the line connecting the light source and the point is larger than the radius of the sphere, the point is not in the shadow of the sphere. This can be expressed as

$$\frac{|\mathbf{L} \times \mathbf{S}|}{|\mathbf{L}|} > R \tag{7}$$

where

$$\mathbf{S} = \text{coordinate}_{\text{center of sphere}} - \text{coordinate}_{\text{pixel}} \tag{8}$$

If a point can not pass the detections for the two situations, that is to say Equation (5) holds, or neither Equation (6) nor Equation (7) holds, the point is in the shadow of the light.

To simplify the calculation, Equation (5) can be expressed as

$$\mathbf{N} \cdot \mathbf{L} < 0 \tag{9}$$

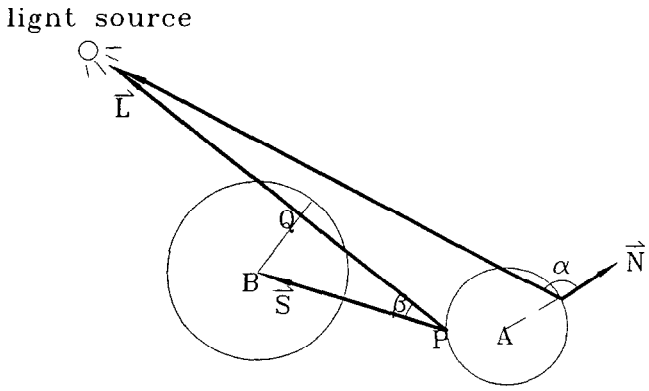

Figure 2. Angle α between a line normal to the sphere surface **N** *and light source direction* **L** *determines if the point is in the shadow of its sphere. If point is in the shadow of sphere* B, β *must be less than 90°, and* BQ *must be less than the radius of sphere* B.

where **N** and **L** can be calculated using Equations (3) and (4). When the line connecting the pixel and the center of another sphere $\mathbf{S}_i$ is calculated, Equations (6) and (7) can be expressed as

$$\mathbf{S}_i \cdot \mathbf{L} > 0 \tag{10}$$

and

$$\frac{|\mathbf{S}_i \times \mathbf{L}|}{|\mathbf{L}|} < R_i \tag{11}$$

respectively, where R is radius of the sphere.

From Equations (9–11), the shadow can be generated.

The calculation of Equations (10) and (11) is time consuming when the molecule is a macromolecule containing thousands of atoms, because for each pixel they must be calculated for all the atoms. We can generate a list which contains the atoms that potentially cast shadows. As shown in Figure 3, the list contains all the spheres which cast shadows on part of the sphere under processing. For all the other spheres in front of the sphere under processing, BC was calculated to check if it is shorter than the radius of sphere B plus DC, where

$$DC = \frac{AE \cdot LC}{LE} \approx \frac{AE \cdot LC}{LA} \tag{12}$$

in which AE is the radius of sphere A,

$$\overrightarrow{LA} = \text{coordinate}_{\text{light source}} - \text{coordinate}_{\text{center of Sphere } A} \tag{13}$$

and

$$\overrightarrow{LC} = |\overrightarrow{\mathbf{LA}}| - |\overrightarrow{\mathbf{CA}}| = |\overrightarrow{\mathbf{LA}}| - \frac{|\overrightarrow{\mathbf{BA}} \times \overrightarrow{\mathbf{LA}}|}{|\mathbf{LA}|} \tag{14}$$

And if so, sphere B should be put in the potential shadowing list for sphere A. Only the spheres in the list are checked by Equations (10) and (11).

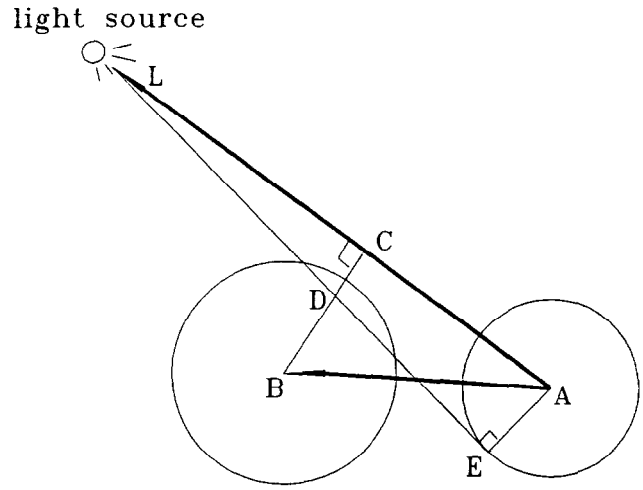

Figure 3. Potential shadowing list is generated by detecting if there is any point on the sphere B *in the shadow of another sphere* A.

GENERAL ALGORITHM

The general flow of the algorithm is as follows:

(1) Transform the molecular model into screen coordinate system, calculate the coordinates X, Y, and Z of the center of all the spheres. Record the radius of the first circle cut from each sphere

$$r_0 = 0$$

and calculate the depth of the circle

$$z = Z - (R^2 - r_i^2)^{\frac{1}{2}} \quad (15)$$

where R is the radius of the sphere. Sort all these circles into a circle list, such that their depth is monotonically increasing.

(2) Take the first circle in the list

(3) For each pixel on the circle, check if it has been drawn on the screen. Do Step 4 for those pixels which have not been drawn, otherwise go to Step 5.

(4) For each light source, check whether the pixel is in its shadow using Equations (9–14). Calculate the intensity of the light according to Equation (2), taking care if the pixel is in the shadows. And then plot the pixel.

(5) Repeat Steps 3 and 4 until all the pixels on the circle have been processed. Here Bresenham's circle algorithm can be used.[7]

(6) Cut and record the radius of the next circle from the sphere just having been processed in Step 2, $r_{i+1} = r_i + 1$. If r_i is R, delete it from the list and jump to Step 8.

(7) Calculate and record the new depth of the circle using Equation (15) and put it at a proper position in the list (sorting according to z).

(8) Repeat Steps 2–7 until the list is empty.

RESULTS

The algorithm proposed in this paper has been implemented in the program MOLMO which has been published earlier.[8] Table 1 shows the performance of the algorithm on IRIS 4D/25 generating full-screen (1280 × 1024) images. The models are in the light of two light sources: one is over the left shoulder at finite distance; the other is coincident with the observer and infinitely far away from the molecule. The CPU times are measured on IRIS 4D/25 running at 25 MHz.

Table 1. Performance

Atoms	Time (CPU sec)	Fraction assigned pixels
617	43.2	0.312
732	56.5	0.351
795	57.8	0.355
861	47.3	0.294
951	40.3	0.228
810	45.3	0.268
1807	86.5	0.321
2325	95.9	0.307
2556	85.9	0.212
2785	139.2	0.403

The results show that the CPU time used to generate the model both depends on the number of atoms and the fraction of the image that the molecule covers.

Compared with Huang's program CONIC, the algorithm presented in this paper is faster for molecules up to about 1000 atoms. It is possible to render the models of small molecules in near real time. Above that number the speed of our algorithm decreased. In Step 5 and Step 3 of the algorithm discussed in the last section, all the circles have to be processed to check if any pixel on them has not been rendered on the screen position even if the circle is completely invisible. For a macro molecule it will take more time to do this. An improvement may be made to test if the circle is completely invisible before checking the pixels on the circle.

Color Plates 1 and 2 present the results of MOLMO. There are two light sources in the sense as described above. The colors are assigned according to the kinds of atoms (blue for nitrogen, red for oxygen, grey for carbon).

CONCLUSIONS

The algorithm for generating space-filling molecular models with shadows has been proposed. The algorithm has no restrictions for the light sources, they can be sport light sources at finite distance or infinitely far away from the molecule. No assumptions have been made in the computation of the image, so the rendering of the model is accurate. The methods for hidden-surface removal, lighting and shadowing have been proposed in detail.

The performance shows that the algorithm is both fast and accurate.

ACKNOWLEDGEMENTS

We thank Dr. Jeremy Hawkes for his help in revising this paper.

REFERENCES

1 Porter, T.K. Spherical Shading. *Comp. Graphics* 1978, **12,** 282

2 Huijsmans, D.P., van Delft, A., and Kuip, C.A.C. WAALSURF: Molecular Graphics on a Personal Computer. *Comput. Graphics* 1987, **11,** 449

3 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

4 Lauher, J.W. Chem-Ray: A Molecular Graphics Program Featuring an Umbra and Penumbra Shadowing Routine. *J. Mol. Graphics* 1990, **8,** 34

5 Palmer, T.C. and Hausher, F.H. Context-free Spheres: A New Method for Rapid CPK Image Generation. *J. Mol. Graphics* 1988, **6,** 149

6 Foley, J.D. and Dam, A. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, 1982, 575

7 Bresenham, J.E. A Linear Algorithm for Incremental Digital Display of Circular Arcs. *Communications of the ACM* 1977, **20,** 100

8 Ai, Z., Li, G., and Wei, Y. MOLMO: Interactive Molecular Graphics on a Personal Computer. *Huaxue Tongbao* 1991, **8,** 59