Conic: A fast renderer for space-filling molecules with shadows

Conrad C. Huang, Eric F. Pettersen, Teri E. Klein, Thomas E. Ferrin, and Robert Langridge

Computer Graphics Laboratory, Department of Pharmaceutical Chemistry, University of California, San Francisco, USA

We present an algorithm for generating images of molecules represented as a set of intersecting opaque spheres. Both perspective and shadows are computed to provide realistic visual cues. Compared to existing programs for generating similar images, our algorithm is both more accurate and several times faster. We present in detail the mathematics used in picture generation, along with examples of the computed images.

Keywords: space filling, shadow, sphere, conic

INTRODUCTION

One of the simplest representations of a molecule is a collection of spherical atoms. While simplistic, this representation is very useful for displaying molecular structure, conformations, volumes, and exposed surfaces. Chemists have long used this representation in the form of plastic Corey-Pauling-Koltun (CPK) models. Therefore it is not surprising, as computer graphics become more popular in the molecular modeling community, that users demand ways of generating space-filling images of molecules on their computer displays.

Many methods have been used to generate space-filling images, ranging from crude overlapping circles to sophisticated ray tracing. One of the first algorithms for generating realistic images was by Porter.[1] (Realistic, in this context, means the generated image is similar in appearance to a photograph of a plastic model.) The Porter algorithm required that the observer and a single light source be coincident and infinitely far away from the molecule. These constraints have two ramifications. First, the generated image projection is orthographic; i.e., spheres in the image do not decrease in size as their distance from the observer increases. Second, there are no shadows in the image. These factors tend to remove depth perception cues, giving the resulting image an unconvincing appearance.

As computers got faster, it became feasible to design algorithms that did not have these constraints. Bacon and Anderson[2] described a tile-based technique for computing images with multiple light sources and shadows. Gwilliam and Max[3] described a resolution-independent technique for computing these images. From the computer science community, ray tracers are available for creating even more sophisticated images with translucent or textured spheres. Unfortunately, all the implementations of these techniques have one common drawback: They are time consuming, either during the preprocessing or the computation phase.

In this paper, we present an algorithm designed specifically for rendering a large collection of possibly intersecting opaque spheres, with an arbitrary number of light sources. The computed image is in perspective and contains analytically computed shadows. Our implementation of this algorithm, called CONIC, is several times faster than the programs mentioned above. In addition, CONIC may be used with Brookhaven Protein Data Bank[4] format files, with no preprocessing, although a configuration file may be supplied to customize the computation.

PREVIOUS WORK

All techniques for generating images of molecules require the definition of several terms. The scene is the collection of molecules to be rendered. The observer is the location or direction from which the scene is viewed. The light sources are the locations or directions from which light shines upon the scene. The image is the picture that the observer sees, and is divided into a rectangular grid. A grid row is called a scanline and a grid element is called a pixel. Standard Cartesian coordinate systems are used for all these techniques, with the z-axis perpendicular to the image.

Porter

The algorithm described by Porter[1] produces images of intersecting shaded spheres. The observer and a single light source are assumed to be infinitely far away in the z-direction. This assumption makes computing the shape and color of spheres quite simple.

The algorithm proceeds sequentially up or down the image by scanline. For each pixel on the scanline, a color and a z-coordinate are computed. Each pixel is initially assigned the same color as the background color and an infinitely distant z-coordinate. As the computation proceeds, a list of active spheres (ones that intersect the current scanline) is maintained by comparing the y-coordinate of the spheres against the y-coordinate of the current scanline. By sorting

Address reprint requests to Dr. Huang at the Computer Graphics Laboratory, Department of Pharmaceutical Chemistry, University of California, San Francisco, CA 94143-0446, USA.
Received 14 August 1990; accepted 1 March 1991

the spheres by their y-coordinates, it is simple and efficient to test when new spheres become active on a given scanline. For each sphere in the active list, the following simultaneous equations are solved for x:

$$r_0^2 = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \tag{1}$$

$$y = y_{\text{scanline}} \tag{2}$$

$$z = z_0 \tag{3}$$

where (x_0, y_0, z_0) are the coordinates of the sphere center, r_0 is the radius of the sphere, and y_{scanline} is the y-coordinate of the current scanline. The solution yields two values for x, $x_{\min}$, and $x_{\max}$, which are the minimum and maximum coordinates at which the sphere intersects the current scanline. For all pixels whose x-coordinate falls between $x_{\min}$ and $x_{\max}$, the actual intersecting coordinate for the sphere and the line of sight through the pixel is computed by replacing Equation (3) with

$$x = x_{\text{pixel}} \tag{4}$$

and solving the simultaneous equations for z. The z-value is compared against the z-coordinate of the pixel. If z is farther, then the sphere is farther away than the previous sphere that intersected the pixel, and the pixel value does not change. Otherwise, the z-coordinate of the pixel is set to z and a new color value for the pixel is computed using Lambert's cosine law as described by Foley and van Dam:[5]

$$color = light_{\text{ambient}} + light_{\text{source}}\left(\frac{z_{\text{pixel}} - z_0}{r_0}\right) \tag{5}$$

The Porter algorithm was designed during a time when computers were relatively slow and memory limitations severe. Thus, it was important to minimize the computational complexity of color calculations as well as the size of data structures associated with spheres. One clever implementation even used the video memory of the raster display subsystem for temporary data storage during the sorting phase of the algorithm.[6] Within these constraints, the images generated using the Porter algorithm were superior to those computed using contemporary algorithms.

Gwilliam and Max

The algorithm described by Gwilliam and Max[3] is used to generate images that include shadows. The observer position is defined relative to the scene to provide approximate perspective, and light sources are assumed to be infinitely distant and to lie in the yz-plane. Unlike the Porter algorithm, the Gwilliam-Max technique first generates resolution-independent decompositions of the scene from the viewpoints of the observer and light sources, and then computes the image based on the decompositions.

The decompositions of the scene consist of a collection of "trapezoids" with straight vertical sides and (possibly) curved top and bottom segments. The spheres are taken sequentially and initially divided into two trapezoids, each with a zero-length vertical side. Obscuring and intersecting spheres are then examined, and may truncate or subdivide the initial trapezoids. The decomposition is first performed for the observer's viewpoint to determine the visible trapezoids of spheres, and then for the light source's viewpoint to determine the lit trapezoids. These decompositions depend only on the number and position of the spheres and are independent of the resolution at which the image is rendered.

After the decomposition preprocessing, the image may be generated by mapping individual trapezoids onto image pixels. An initial color value for a pixel is computed by placing a dim light source coincident with the observer and applying Lambert's cosine law. The pixel is then checked against the lit trapezoids to determine whether it is in shadow. If not, then the color contribution of the main light source is added to the pixel color value. This phase of the algorithm is dependent on the resolution of the image.

The Gwilliam-Max algorithm contains several approximations (e.g., circles and circular arcs instead of ellipses) and constrains the light source to be in the yz-plane. These limitations, as well as the preprocessing requirements, reduces its suitability in an interactive modeling environment.

Bacon and Anderson

The algorithm described by Bacon and Anderson[2] also generates images containing shadows. The observer's position is defined relative to the scene, with one infinitely distant light source and another at the observer's position. This algorithm handles not only spheres, but also triangles. Therefore the techniques used are not optimized for scenes containing only spheres.

The heart of the algorithm is the division of the image into a set of rectangular tiles. For each tile, the set of spheres is computed that project onto it. The advantage of using this division is that there are relatively few spheres associated with each tile, making the tile image faster to calculate.

A tile image is computed using a priority list combined with z-buffer. The spheres associated with the tile are sorted by their distance from the observer. Then, for each pixel in the tile, the list of spheres is searched to find the closest sphere that maps onto the pixel. Once the coordinates associated with the pixels are found, a similar calculation is done relative to the infinitely distant light source to determine whether the pixel is in shadow. Finally, the shading computation is done as described in Foley and van Dam.[5]

The implementation of this algorithm by Bacon and Anderson, called raster3d, produces good quality images. There are two shortcomings of this system, however. First, it assumes that the silhouettes of spheres are circles rather than the conic sections. Second, it takes over 8 minutes to compute most images (for scenes containing between 3 and 5000 spheres) on a reasonably fast computer, a Silicon Graphics IRIS 4D/80GT.

Ray Tracing

Ray tracing is a computer graphics technique for creating photorealistic images. The color of each pixel in the image is calculated by tracing backwards along the light ray that passed through the pixel and reached the observer. A detailed description of ray tracing is beyond the scope of this paper. A good treatment is found in Glassner.[7] The images generated using this technique range from striking to spectacular. The main drawback is the great amount of processing time required to compute the images.

PRESENT ALGORITHM

The algorithm we present is for computing the image of a scene containing only opaque spheres, and closely parallels Porter's algorithm except that we remove the constraint of having the observer and a single light source be coincident and infinitely distant from the scene. Figure 1 shows the scenario that our algorithm is designed to handle. The image is generated by projecting the scene into the plane of the image and computing the color of each pixel. Placing the observer a finite distance from the scene rather than infinitely far away leads to one major difference. When a sphere is projected into the image, its silhouette is a circle of the same radius in the infinite case, but is a conic section, generally an ellipse, in the finite case. Allowing light sources that are not coincident with the observer leads to shadows in the computed image. These differences result in greater complexity in computing the image. We present the mathematics for generating the image correctly, as well as some techniques for reducing the computation time and memory requirements.

General Algorithm

The general flow of our algorithm is as follows:

(1) Compute the rotation matrix that transforms the observer's position onto the positive z-axis, and apply it to all sphere center coordinates.
(2) For each light source that is infinitely distant from the scene, find a plane perpendicular to the light direction. Divide the plane into a rectangular grid and, for each grid element, find the set of spheres that project into the element in the direction of the light source.
(3) Sort the spheres into a sphere list, such that the y-coordinate of the bottom edge of their silhouettes, as projected toward the viewer, is monotonically increasing.
(4) This is the prologue of the image-computation loop, which proceeds scanline by scanline. We start with the bottommost scanline and form an empty active list.
(5) Update the active list to include exactly the set of spheres whose projected silhouettes intersect the current scanline. First, we remove from the active list any spheres that no longer intersect the current scanline, i.e., whose projected upper edge is below the current scanline. Then, we add to the active list any spheres in the sphere list whose projected lower edge intersects the current scanline. Because the sphere list is sorted by the lower projected edge, the addition phase is done simply and quickly by moving spheres from the head of the sphere list onto the active list until we come to a sphere whose lower edge lies above the current scanline.
(6) For each pixel on the current scanline, compute the intersections of the line defined by the observer's and pixel positions and the spheres on the active list. The intersection nearest to the observer is used to define the color of the pixel. We begin by marking all pixels as unassigned; i.e., they intersect no spheres. Then, for each sphere on the active list, we take the range of pixels on the current scanline that falls within the silhouette of the sphere and compute intersection points for them. If the newly computed intersection point for an assigned pixel has a more distant z-coordinate than the assigned intersection point, we discard it; otherwise, we assign the sphere and intersection point to the pixel. After all spheres on the active list have been processed, the pixels on the current scanline either are unassigned or have coordinates associated with them.
(7) For each pixel on the current scanline, calculate its color. If a pixel is unassigned, give it the background color. Otherwise, compute the color at the assigned pixel coordinate by summing the contributions from ambient light and all light sources, taking care to handle shadows properly.
(8) Repeat steps 5 through 7 for the scanline above until we reach the topmost scanline in the image.

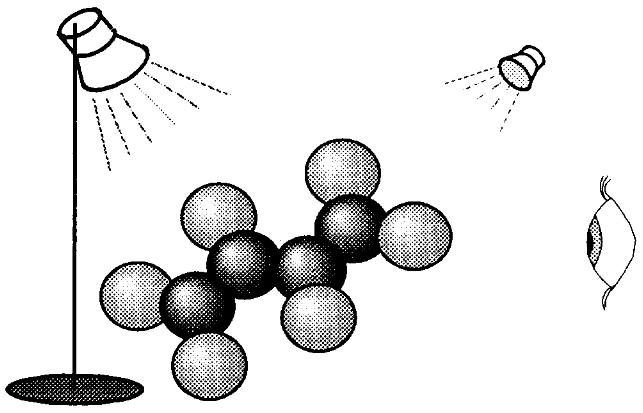
Figure 1. Generated image is based on the observer being a finite distance from the scene and there being multiple light sources

Conic Sections

In Steps 3 and 6 above, we need to compute the silhouette of a sphere when projected into the image plane, as shown in Figure 2. The silhouette is the intersection of the image plane and a right circular cone with its apex at the observer position and its axis passing through the center of the sphere. The equation for this cone may be derived in two steps. First, consider a cone whose apex is at the origin and whose axis coincides with the positive z-axis. The equation for this cone is

$$x^2 + y^2 = (nz)^2 \qquad (6)$$

Second, the transformation matrix that rotates the positive z-axis onto an arbitrary axis is

$$T = \begin{pmatrix} \cos\theta\cos\phi & -\sin\theta & -\cos\theta\sin\phi \\ \sin\theta\cos\phi & \cos\theta & -\sin\theta\sin\phi \\ \sin\phi & 0 & \cos\phi \end{pmatrix} \qquad (7)$$

as described in Foley and van Dam.[8] The angles θ and ϕ

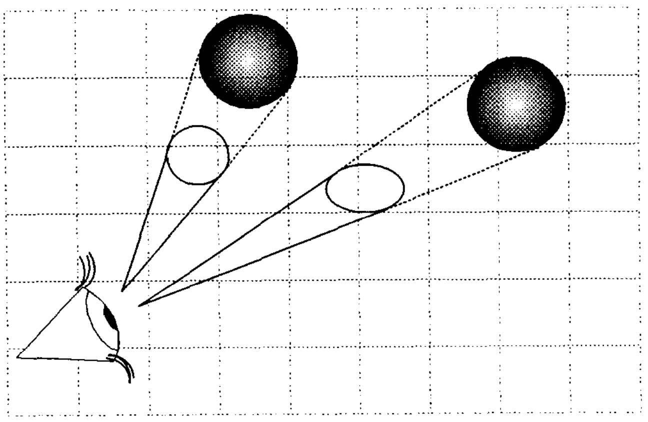

Figure 2. Spheres project as conic sections in the image plane, which is denoted by the grid

are the longitude and latitude of the arbitrary axis. Applying the matrix, we get

$$\begin{aligned} x_{\text{xform}} &= x\cos\theta\cos\phi + y\sin\theta\cos\phi + z\sin\phi \\ y_{\text{xform}} &= -x\sin\theta + y\cos\theta \\ z_{\text{xform}} &= -x\cos\theta\sin\phi - y\sin\theta\sin\phi + z\cos\phi \end{aligned} \tag{8}$$

Combining Equations (6) and (8), we arrive at the classic quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 \tag{9}$$

with

$$\begin{aligned} A &= \cos^2\theta\cos^2\phi + \sin^2\theta - n^2\cos^2\theta\sin^2\phi \\ B &= -2\cos\theta\sin\theta\sin^2\phi(1 + n^2) \\ C &= \sin^2\theta\cos^2\phi + \cos^2\theta - n^2\sin^2\theta\sin^2\phi \\ D &= 2z\cos\theta\cos\phi\sin\phi(1 + n^2) \\ E &= 2z\sin\theta\cos\phi\sin\phi(1 + n^2) \\ F &= z^2(\sin^2\phi - n^2\cos^2\phi) \end{aligned}$$

Equation (9) is a general description of right circular cones having their apexes at the origin, and has three parameters θ, ϕ, and n. For our case of sphere and observer, we can simply replace (x, y, z) with $(x - x_{\text{observer}}, y - y_{\text{observer}}, z - z_{\text{observer}})$ to translate the observer to the origin. If our translated sphere has Cartesian coordinates $(x_{\text{sphere}}, y_{\text{sphere}}, z_{\text{sphere}})$ and radius r_{sphere}, the parameters for Equation (9) are

$$\theta = \tan^{-1}\frac{y_{\text{sphere}}}{x_{\text{sphere}}} \tag{10}$$

$$\phi = \cos^{-1}\frac{z_{\text{sphere}}}{(x^2_{\text{sphere}} + y^2_{\text{sphere}} + z^2_{\text{sphere}})^{1/2}} \tag{11}$$

$$n = \frac{r_{\text{sphere}}}{(x^2_{\text{sphere}} + y^2_{\text{sphere}} + z^2_{\text{sphere}} - r^2_{\text{sphere}})^{1/2}} \tag{12}$$

Once we know the equation for the cone, computing the projected silhouette is a simple matter. The image lies in the plane

$$z = z_{\text{image}} \tag{13}$$

To compute the top and bottom edges of the silhouette of a sphere, for Steps 3 and 5 in our algorithm, we use the fact that there can only be a single value for x at the edges. Thus, if we treat Equation (9) as a quadratic in x with y as a parameter,

$$Ax^2 + (By + D)x + (Cy^2 + Ey + F) = 0 \tag{14}$$

we know that the discriminant must be zero, i.e.,

$$(By + D)^2 - 4A(Cy^2 + Ey + F) = 0 \tag{15}$$

Solving Equation (15) for y, we obtain the top and bottom edges. To compute the left and right edges of the silhouette at a particular scanline for Step 6, we simply set

$$y = y_{\text{scanline}} \tag{16}$$

and solve Equations (9), (13), and (16) simultaneously for x.

After we have determined the range of pixels that fall within the silhouette, we need to compute the intersection between the sphere and the line that passes through the observer and the pixel positions. The sphere is described by

$$(x - x_{\text{sphere}})^2 + (y - y_{\text{sphere}})^2 + (z - z_{\text{sphere}})^2 = r^2_{\text{sphere}} \tag{17}$$

and the line is described parametrically as

$$\begin{aligned} x &= (x_{\text{pixel}} - x_{\text{observer}})t + x_{\text{observer}} \\ y &= (y_{\text{pixel}} - y_{\text{observer}})t + y_{\text{observer}} \\ z &= (z_{\text{pixel}} - z_{\text{observer}})t + z_{\text{observer}} \end{aligned} \tag{18}$$

Combining Equations (17) and (18), we can solve for t. The smallest positive value of t corresponds to the desired intersection point and substituting back into Equation (18) yields its coordinates.

Armed with these equations, we can proceed through the first 7 steps of the algorithm. The simplest method for Step 6 is to compute the intersection points for all the pixels for all the active spheres and to use a z-buffer to handle all hidden-surface removal. Unfortunately, this can be time consuming because solving Equations (17) and (18) requires a square root calculation. An optimization we use to reduce CPU time is to compute only three intersection points per sphere initially. Two of the computed points are the left and right edges; the third is the intersection point that approaches closest to the observer for the entire range of pixels. If the closest intersection point maps to a pixel that is assigned to a particular sphere and has a closer z-coordinate, and if one of the edge points maps to a pixel that is assigned to the same sphere and also has a closer z-coordinate, then we do not bother to compute any intersection points for pixels between that edge and the closest pixel, because the new intersection points cannot possibly have closer z-coordinates than the assigned ones. This optimization, coupled with sorting the active list by the z-coordinates of the spheres, can greatly reduce the number of intersection computations since spheres not visible to the observer are quickly eliminated.

Lighting Computation

The lighting model used in our algorithm is that described in Foley and van Dam[5] and is shown in Figure 3. The equation that we use for computing the pixel colors in this lighting model is†

$$pixel_{\text{color}} = light_{\text{ambient}} k_a + \sum light_{\text{source}}[k_d(\bar{L} \cdot \bar{N}) + k_s(\bar{R} \cdot \bar{V})^n] \qquad (19)$$

where

k_a = ambient-reflection coefficient
k_d = diffuse-reflection coefficient
k_s = specular-reflection coefficient
n = Phong shading model parameter

Here k_a, k_d, k_s, and n are properties of the surface and may vary from sphere to sphere.

We know the sphere to which each pixel is assigned and the intersection coordinate. We find that

$$\begin{aligned}
\bar{L} &= coordinate_{\text{light}} - coordinate_{\text{pixel}} \\
\bar{N} &= coordinate_{\text{pixel}} - center_{\text{sphere}} \\
\bar{R} &= \bar{N} - (\bar{L} - (\bar{L} \cdot \bar{N})\bar{N}) \\
\bar{V} &= coordinate_{\text{observer}} - coordinate_{\text{pixel}}
\end{aligned} \qquad (20)$$

If there are no shadows in the image, then simply applying Equations (19) and (20) gives the pixel colors. However, because some spheres cast shadows on other spheres, Equation (19) should actually be

$$pixel_{\text{color}} = light_{\text{ambient}} k_a + \sum \begin{cases} 0 \text{ if in shadow} \\ light_{\text{source}}[k_d(\bar{L} \cdot \bar{N}) + k_s(\bar{R} \cdot \bar{V})^n] \text{ otherwise} \end{cases} \qquad (21)$$

The general procedure for determining whether the assigned coordinate of a pixel is in shadow from a particular light source is to construct and check against a blocker list. The blocker list is the set of spheres that potentially cast shadows on the pixel coordinate. For an infinitely distant light source, we can use the grid constructed in Step 2 of the algorithm to quickly determine the blocker list. For a light source at finite distance, we need to check all spheres that are closer to the light source than the sphere assigned to the pixel. To determine if a test sphere potentially casts a shadow, we use the light source at finite distance as the angle vertex and compare the angle between the test sphere and the assigned coordinate against the sum of the angles subtended by the two spheres. If the former is less, then the test sphere potentially casts a shadow on the pixel coordinate. Once the blocker list is constructed, we check whether the line passing through the pixel coordinate in the direction of the light source intersects any sphere in the list. If so, then the pixel coordinate is in shadow, and the light source does not contribute to the color of the pixel.

†Equation (19) is slightly different to that in Foley and van Dam in that the intensity of a finite light source does not fall off with distance. This causes a slight difference in the computed image but reduces computation time.

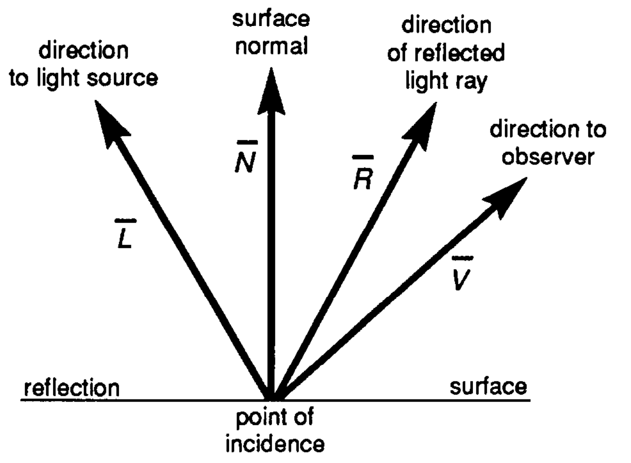

Figure 3. Lighting model used for computing pixel colors from Foley and van Dam

In the infinitely distant light source case, the grid is an efficient means of determining blocker lists because it only needs to be constructed once, and most grid elements are eventually used because they normally map to one or more image pixels. In the case of a light source at finite distance, the blocker list for a sphere is constructed when the color of its first pixel is computed. It is expensive both in computation time and memory usage to construct the finite light blocker list for all spheres, and only those for visible spheres are used. For large molecules, a substantial portion of the spheres are obscured, making it inefficient to construct the complete lists.

Discussion

The algorithm presented above is designed to generate images from scenes composed strictly of opaque spheres. The observer may be arbitrarily positioned at any finite distance from the scene. Any number of light sources may be arbitrarily positioned a finite or an infinite distance from the scene. The computed image has the proper perspective, highlights, and shadows to enhance the image's realism. No approximations that affect image quality have been made. The techniques used to compute the pixel intersection coordinates and pixel color are very similar to those used in ray tracing. In fact, our algorithm produces the same images as that of a ray tracer that does not compute reflections, but in much less time.

RESULTS

CONIC, an implementation of our algorithm on a Silicon Graphics Iris 4D/80GT, has been in operation for over a year at the UCSF Computer Graphics Laboratory. The program takes as input a Brookhaven Protein Data Bank[4] file and, optionally, a configuration file. The PDB file contains the atom coordinates and, possibly, properties such as radius and color. The configuration file contains display options shown in Table 1. Detailed descriptions of execution and configuration options for CONIC may be found in the UNIX manual page in the appendix.

Table 1. Display options available in CONIC

Option	Default
Default input PDB file	None
Default image output file	None
Default atom property file	System version
Antialias mode	None
Field of view	30°
Default sphere properties	$k_s = 0.5$, $k_d = 0.25$, $n = 8$
Ambient light color	White
Observer light color	White
Background color	Black
Cone light source	None
Spot light source	None
Infinite light source	One over right shoulder
Finite light source	None

Table 2 shows 10 proteins, computation times to generate full-screen (1280 × 1024) images without antialiasing, the number of atoms, and the fraction of pixels in the image that were assigned to spheres. All times are from an Iris 4D/80GT (MIPS R2000 chip set running at 16.67 MHz). The results show that computation time is less dependent on the number of atoms than on the fraction of the image that the atoms cover. Thus the performance of CONIC varies greatly in the shape and orientation of the molecules rendered.

Table 3 compares the computation time of antialiased full-screen images for CONIC and several other programs. Program raster3d is by Bacon and Anderson. Program rayshade is an excellent general ray tracer written by Craig Kolb.[9] Program cpk is a fast implementation of the Porter algorithm by Thomas Ferrin.[10]

Although the image generation time of CONIC is too long to support real-time manipulation, it is short enough that CONIC is frequently used in conjunction with MidasPlus.[11,12] Rotation and translation of molecules may be done quickly in MidasPlus using wire-frame models, and space-filling images of specific conformations and views are generated using CONIC via the PDBRUN command of MidasPlus, which sends the transformed coordinates of the displayed molecules as input to external programs.

Table 3. Computation time, in CPU seconds, for various molecules and programs

	conic	raster3d	rayshade*	cpk**
5rxn	128.4	569.	784.4	37.2
2lhb	148.1	589.	852.0	50.0
1bds	152.3	567.	869.0	33.7

*rayshade was constrained to cast no reflection or refraction rays. Thus, the above times are for producing images of approximately the same quality as CONIC.

**cpk uses a color map instead of full 24-bit color. Thus the program cannot properly compute the color of a pixel that falls on the intersection of two spheres with different colors.

Color Plates 1–4 present images computed using CONIC and MidasPlus. Color Plates 1 and 2 show trimethoprim, an antibacterial drug, with different material property and lighting conditions. Color Plates 3 and 4 show wild and mutant forms of trichosanthin (Compound-*Q*),[13] a protein being tested for treatment of AIDS and its related conditions. By using a spotlight, it is easier for the viewer to discern the difference between two structures that may differ by only a mutation or for highlighting a particular region of interest.

CONCLUSION

An algorithm for generating images from scenes consisting of spheres is presented. The observer is a finite distance from the scene and any number of arbitrarily positioned light sources may be specified. The images, computed using equations derived for conic sections and shaded with both diffuse and specular reflection, display the proper perspective and shadows essential for a realistic picture. Used in conjunction with MidasPlus, CONIC has provided a fast and easy means of creating space-filling images of molecules.

ACKNOWLEDGEMENTS

We appreciate the use of the model of trichosanthin from Jon Robertus at the University of Texas and the mutated forms of trichosanthin from Genelabs in Redwood City. The

Table 2. CONIC performance

Protein	PDB[4] Name	Time (CPU sec)	Atoms	Fraction assigned pixels
Ribonuclease A	5rsa	51.4	1777	0.273
Trypsin inhibitor	5pti	52.5	874	0.303
Cytochrome C	351c	56.1	712	0.356
Cytochrome C	451c	56.9	718	0.362
Rubredoxin	5rxn	60.0	873	0.381
Myoglobin	1mb5	64.0	2303	0.338
Rubredoxin	4rxn	65.2	884	0.423
BDS-I	1bds	70.5	629	0.413
Hemoglobin V	2lhb	72.3	2620	0.407
Trypsin	1ntp	74.5	3075	0.376

model is based on ricin *A*-chain coordinates described by Collins, et al.[13]

We thank Julie Newdoll, an early user of the raster3d program in our laboratory, for suggesting the need of a faster and more flexible program, for extensively testing early versions of CONIC, and for suggesting useful new features.

This work was supported by National Institutes of Health, National Center for Research Resources, grant RR-01081. The conic program is available as part of the MidasPlus software package.[14]

REFERENCES

1 Porter, T.K. Spherical Shading. *Comp. Graphics* (1978) **12**, 282
2 Bacon, D. and Anderson, W.F. A fast algorithm for rendering space-filling molecule pictures. *J. Mol. Graphics* (1988) **6**, 219
3 Gwilliam, M. and Max, N. Atoms with shadows—An area-based algorithm for cast shadows on space-filling molecular models. *J. Mol. Graphics* (1989) **7**, 54
4 Abola, E.E., Bernstein, F.C., Bryant, S.H., Koetzle, T.F. and Weng, J. In: *Crystallographic Databases—Information Content, Software Systems, Scientific Applications*. (F.H. Allen, G. Bergerhoff, and R. Seivers, Eds.) Data Commission Intl. Union Crystallogr., Cambridge, 1987, 107–132
5 Foley, J.D. and Dam, A. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, 1982, 575
6 Porter, T.K. The Shaded Surface Display of Large Molecules. *Comp. Graphics* (1979) **13**, 234
7 *An Introduction to Ray Tracing*. (A.S. Glassner, Ed.) Academic Press, New York 1989
8 Foley, J.D. and Dam, A. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, Reading, 1982, 279
9 Kolb, C. Rayshade *UNIX Manual Page*. Usenet, 1988
10 Ferrin, T.E. *CPK User's Guide*. Univ. of California, San Francisco, 1979
11 Ferrin, T.E., Huang, C.C., Jarvis, L.E. and Langridge, R. The MIDAS display system. *J. Mol. Graphics* (1988) **6**, 13
12 *UCSF MidasPlus User's Manual*. Univ. of California, San Francisco, 1989
13 Collins, E.J., Robertus, J.D., LoPresti, M., Stone, K.L., Williams, K.R., Wu, P., Hwang, K. and Piatak, M. Primary amino acid sequence of α-trichosanthin and molecular models for abrin A-chain and α-trichosanthin. *J. Biol. Chem.* (1990) **265**, 8665–8669
14 *MIDAS Software Distribution*. University of California, San Francisco