The application of fractal clustering to efficient molecular ray tracing on low-cost computers

David T. Jones

Biomolecular Structure and Modelling Unit, Department of Biochemistry and Molecular Biology, University College, London, UK

Ray tracing is a powerful, and highly computer intensive means for generating high-quality molecular displays. A variety of simple, yet effective optimization strategies are described that allow large molecular models to be ray traced on microcomputers and low-cost desktop workstations. In particular, the method of fractal clustering provides a time and space-efficient means for spatially subdividing the molecular scene into a hierarchy of spherical bounding volumes, permitting ray–atom intersections to be determined by a form of binary search. An implementation of the algorithms, MolRay, is described which demonstrates that large structures may be ray traced in a reasonable time on a PC or small Unix workstation. Images generated by PC and Unix versions of MolRay are shown.

Keywords: ray tracing, optimization, molecular modeling, microcomputers, raster graphics, visualization, fractal clustering

INTRODUCTION

Many different display techniques are currently used to visualize complex molecules. Though wire-frame and simple ball-and-stick displays are popular for quickly navigating around a molecule during, for example, protein modeling, more detailed solid model styles are preferred for final displays and publications, the CPK style being one of the most popular.

Ray tracing provides the means for generating the most visually attractive of all molecular displays, as shown clearly by Palmer[1] and Lauher,[2] for example. As a graphics technique, ray tracing was first described by Appel[3] though the more complete algorithm now in use is attributable to Whitted.[4] The algorithm constructs a scene by following the path of light rays in reverse: tracing the path back from the observer position, through the viewing screen onto the objects in the scene, and finally back to the light sources. Detailed optical effects, such as shadows, reflection (diffuse and specular), and even refraction, are accurately simulated by ray tracing, and the optical characteristics of object surfaces are under complete control. These methods are generally known as "photorealistic," though this is, of course, a rather questionable term to apply to the rendering of molecules. The provision of 'real-looking' molecular displays does, however, greatly enhance the ease with which complex models can be comprehended. Unfortunately, ray tracing is a very tedious process, involving an enormous amount of computing time to generate even the simplest scenes.

The possibility of generating ray-traced molecular displays on very modest single-processor computer systems is considered here. A modest computer system as considered here may be represented by anything from a 12-MHz 80286-based PC-compatible microcomputer (with a VGA display) to a general purpose desktop workstation, such as the Sun SPARCstation IPC. Obviously a color graphics display is an almost essential requirement, though quite reasonable halftone output is possible to laser printers, for example. Again, compromises must be made for these modest systems. Ideal ray-tracing frame buffers provide at least 24 bitplanes (each pixel having a free choice of 16 777 216 colors), but the best that can be expected from a modest system is an 8-bitplane frame buffer where each pixel can assume any of 256 colors chosen from a palette of, say, 262 144 colors (for PC VGA graphics). There is no harm in supporting optional (sadly expensive) multibitplane devices, however.

The program described here was designed to generate fully ray-traced molecular images on anything from the aforementioned VGA-equipped 80286-based PC (ideally with a floating-point coprocessor), to an entry-level color workstation like the Sun SPARCstation IPC. Given that molecular displays generally comprise more than 500 atoms (and very often more than 5000), algorithmic efficiency was a prime concern.

No attempt at describing the basic principles of ray tracing will be presented here. For those unfamiliar with the technique, Glassner[5] provides an excellent introduction. The program described here uses a standard approach to the core ray-tracing process, using recursion to handle multiple reflections from mirrored surfaces as usual.

Address reprint requests to Mr. Jones at the Biomolecular Structure and Modelling Unit, Department of Biochemistry and Molecular Biology, University College, Gower Street, London, WC1E 6BT, UK.
Received 7 February 1991; accepted 12 March 1991

© 1991 Butterworth–Heinemann

ACCELERATING THE RAY TRACING ALGORITHM

Given a basic ray-tracing framework suitable for a simple scene comprising a single sphere, for example, how can it be extended to handle relatively complex molecular scenes? The simplest approach is to construct a linear list of atomic spheres in the scene, along with a linear list of light sources illuminating the scene. Many basic ray-tracing programs use this approach, and for relatively simple scenes it is quite adequate. Profiling a simple ray tracer when generating even a simple molecular display indicates that as much as 90 % of the computing time is spent simply calculating the line–sphere (ray–object) intersections, the great majority of which are found to be imaginary (i.e., the line and sphere do not intersect). Clearly, to accelerate the ray-tracing process, the amount of time spent calculating these intersections must be reduced.

The ray-tracing process may be accelerated in several ways. By far the most effective way is simply not to do it at all. The shadow-generation method of Williams,[6] which uses two passes through a standard z-buffer algorithm (one for the viewer, which identifies visible pixels, and one for the light source, which identifies pixels that are visible and shadowed), can produce very impressive results. However, if we wish to simulate the more intricate visual effects mentioned earlier without developing separate simulations for each one, we are left with ray-tracing as the only viable means. In addition, when many of the approximate sphere-shading and shadowing algorithms are applied to molecular graphics, they do not take proper account of the effects of perspective, which are handled naturally by ray tracing.

If we consider that the total time taken in computing these ray–sphere intersections is given by NT, where N is the total number of line–sphere intersection tests actually performed (directly proportional to the number of spheres in the scene, for the simple ray tracer), and T is the time taken to perform one of them, it is clear that halving either N or T, for example, will double the speed of the ray-tracing program. Ideally we should like to reduce both factors.

Tackling T would appear to be the simplest approach. Unfortunately, testing for a line–sphere intersection is a relatively straightforward procedure, and there is little scope for radically improving on performance here. Haines,[7] however, describes an improved line–sphere intersection test using a geometric rather than analytic approach, which appears to be 50 % faster.Though this represents a worthwhile time saving, even with this improvement, ray tracing a scene comprising as few as 500 atoms is simply not practicable.

REDUCING THE NUMBER OF INTERSECTION TESTS

Clearly, the only way of significantly increasing the efficiency of ray tracing is to cut down on the number of ray intersection calculations. A standard way to achieve this is to break the scene down into a hierarchical set of bounding volumes (which can be spheres, ellipsoids, boxes, polyhedra, or even more complex structures). At the root of this tree structure the bounding volume encloses all the objects in the scene; at the leaf level, the bounding volumes are in fact the objects themselves. If no intersection is found between a ray and a bounding volume, then no intersection is possible between the ray and any objects contained in this volume. Roughly speaking, this reduces the ray intersection problem from $O(n)$ to $O(\log n)$ for the case of a balanced binary tree and nonrecursive ray tracing. For recursive ray tracing, the order of the problem increases with the allowed depth of recursion, making effective volume hierarchies essential. With care, these hierarchies may be constructed manually, generally with the help of a structure-building program, but for molecular ray tracing such bounding volume hierarchies are generally not provided with the atomic coordinate data. It is, perhaps, interesting to note that for protein data there exists a 'natural' object hierarchy in the division of a protein structure into subunits, domains, subdomains, residues, and finally, atoms. The method described here, however, simply considers a molecule as a set of atoms, and rapidly builds an effective bounding volume hierarchy for a given set of atomic coordinates. It may well be possible to make use of the natural protein structure hierarchy, but this possibility will not be further considered here.

FRACTAL CLUSTERING

Automatically generating a bounding volume hierarchy for a collection of objects is akin to the statistical technique of cluster analysis, whereby items are sorted into 'clusters' depending on the distances between items. Traditional clustering procedures are expensive in both time and space. Every interobject distance must be known before clustering can proceed, requiring, for example, 124 750 distance calculations and storage for these values in the case of a scene comprising 500 objects (a fairly small biomolecule). A better clustering algorithm for large data sets has been proposed by Zupan,[8] which is known as "3-distance" or "fractal clustering." This algorithm begins with a tree comprising just two items. Additional items are added to this tree one at a time, and may be inserted into the tree at any point depending on the 3-distance criterion. Thus tree growth at points other than at the leaves of the tree is similar to the growth pattern of 'fractal trees,' hence the pseudonym of fractal clustering.

The 3-distance criterion is illustrated in Figure 1 and is described in detail by Zupan.[8] In simple terms, the 3-distance criterion affords a straightforward way of locating the 'correct' location of an item in a pre-existing tree structure. Starting at the root of a binary tree, there are 3 possible courses of action: stop, go left, or go right. To make the correct decision, 3 distances are calculated: the distance between the new item and the left cluster (D_L), the distance between the new item and the right cluster (D_R), and the distance between the left cluster and the right cluster (D_3). To speed up the calculation of cluster distances, the centroid of the cluster beneath is stored at each tree node (the actual object centroids are stored, in the case of leaf nodes). If $D_3 < D_L$ and $D_3 < D_R$, the stopping condition is reached, and the new object is inserted at the root of the tree. (It is presumed that the new object is insufficiently close to any of the existing clusters and so is permitted to start its own cluster.) If the third distance criterion is not satisfied, the

object takes the left branch if $D_L < D_R$ or the right branch otherwise. The process continues until either the third distance criterion is satisfied (the object being inserted at this point) or the object arrives at the leaf level of the tree, in which case it is joined to the closest leaf node.

The tree structure generated by this procedure is generally not optimal. Further improvement may be achieved by softening the third distance criterion, which is achieved by increasing D_3 by a small amount: this encourages new objects to join existing clusters rather than to start new clusters. Further improvement may be achieved by iterating the tree. This involves removing objects from the tree one-by-one and re-inserting them (taking care to recompute the node centroids as necessary). The point is that an object is more likely to find its natural cluster position when the tree is almost complete than it is in the early stages of tree construction when few clusters will have been formed. It is easy to test whether an object is in its natural cluster position by performing the 3-distance procedure again, after the object has been inserted into the tree. If the object is correctly located, then the search should succeed in finding the object even after other objects have been added. Failure to rediscover the object's location indicates that the object is wrongly positioned and should be iterated. Zupan[8] covers these, and other tree optimizations in detail.

In general, the raw 3-distance tree has been found to be quite satisfactory for the purpose of accelerating the ray-tracing process, though the time taken to optimize the tree structure is not significant compared to the overall ray-tracing time. Using the fractal clustering algorithm on a Silicon Graphics Personal Iris 4D/20, 8000 atoms may be hierarchically clustered into bounding volumes in just 7 CPU seconds. The program described here uses simple spherical bounding volumes, defined at each node in the 3-distance tree. Each bounding sphere is centred on the node's cluster centroid, and has a radius just sufficient to completely enclose all the objects in the cluster.

Figure 2 shows the conceptual outline of the final tree structure. To see how this structure cuts down on the required number of intersection tests, a typical tree-based

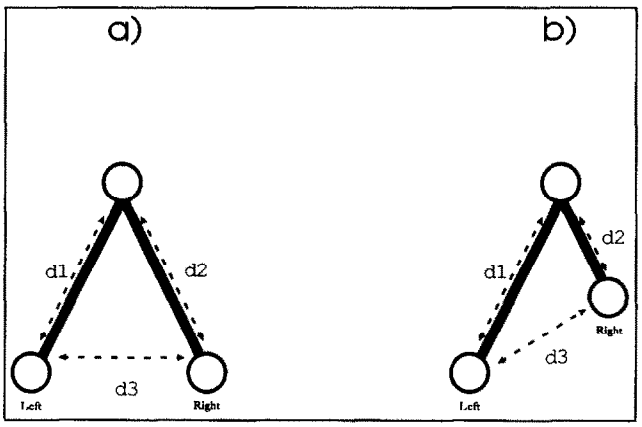

Figure 1. Three-distance criterion for the nodes of a simple two-branch tree with branches of equal length (a) and unequal length (b). In this case, the branch lengths have a physical meaning: essentially, the distances between atoms

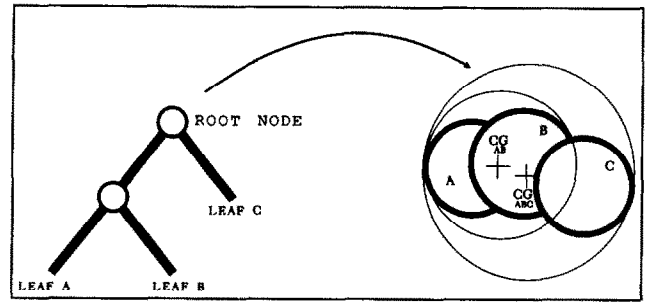

Figure 2. Example of a bounding volume hierarchy for three atomic spheres, as produced by fractal clustering. The root node holds the centroid of the entire cluster ABC *and describes a sphere just sufficient to envelope the three spheres. Likewise, the* AB *node holds the centroid of the cluster* AB *and describes a sphere just sufficient to envelope these two spheres*

intersection test should be considered. Starting at the root of the tree, the bounding sphere here completely encloses all objects in the scene. If the ray does not intersect this sphere it is obvious that it cannot intersect any sphere in the scene. Given that the ray does in fact intersect the root bounding sphere, the testing procedure descends one level in the tree (right and left) and tests for an intersection with the two bounding spheres ("children" of the root node). If the ray is not found to intersect either of these spheres, testing can terminate as it is clear that the ray cannot intersect any bounding spheres further down the tree. The entire object tree is quickly traversed via simple recursion. For secondary shadow rays it is sufficient to know that any sphere blocks the path to the light source rather that worrying about which is the closest intersection, this recursive search can terminate therefore as soon as an intersection is found to any leaf node.

OTHER OPTIMIZATION STRATEGIES

These optimization strategies are generally applicable in that they generate correct images under all circumstances. Certain heuristic optimizations can be applied to molecular ray tracing that offer significant speed gains at the expense of risking rendering errors. Two optional optimization methods are used: automatic framing and tree pruning. Both of these methods take advantage of the fact that images are generated in several passes, by increasing the pixel resolution by a factor of 2 for each successive pass. The initial passes generate low-resolution images, but these images are quite suitable for providing a preview of the final full-resolution image. These low-resolution images are, in fact, sampled from the full-resolution image; in the first pass the color of each 8 × 8 pixel block (x:0–7, y:0–7) would assume the color of the pixel at (x:0, y:0) in the final image. Care is taken that successive passes do not waste time recalculating pixel colors that have been determined already. Given that the last pass ends up calculating all the pixels corresponding to the odd numbered coordinates, it ends up rendering 75 % of the total pixels.

Automatic framing uses the low-resolution sampling to calculate a frame around the image, thus avoiding the calculation of empty regions where no ray–sphere intersections will be expected. This assumes that the molecule does not occupy the entire screen, in which case no benefit would be gained from this method.

A more useful heuristic optimization is tree pruning. Molecular models differ somewhat from the typical ray-tracing scene in that many of the objects are wholly obscured by others. Medium to large protein molecules, for example, have a large number of completely buried atoms that contribute nothing to the final image, and much to the overall burden of ray–sphere intersection calculation. It is clearly advantageous to exclude these buried atoms early on in the ray-tracing process. To accomplish this, each node in the object tree holds a number of "visit flags." When a ray is found to intersect the bounding sphere of a particular node, the visit flag is set, indicating that at some point rays have reached this particular node in the tree. Later passes can use these flags to avoid branches of the tree offering no probable ray–sphere intersections, the ray-tracing program thus 'learns' to avoid futile intersection tests. Of course there is a danger that the information gained from earlier low-resolution passes will no longer be valid for higher resolution passes, although for molecular displays, it has been found that pruning the object tree before the last pass has virtually no effect on the final image while significantly accelerating the ray-tracing process. It is important, however, to differentiate between reverse shadow rays and eye rays in the pruning process. As shadows tend to occupy relatively little screen area, tree pruning for shadow ray intersections should occur at a higher resolution than for eye rays.

A further refinement of tree pruning takes into consideration the fact that there is a strong coherence between screen pixel coordinates and the set of spheres that may be intersected by the related eye rays. Rather than implementing a flag that indicates whether any eye rays have reached a particular node in the object tree, different flags are used for different zones of the viewing screen. For machines with limited memory, the screen is divided into a 4×3 grid, giving 12 eye ray intersection flags; otherwise a 6×5 grid is used.

IMPLEMENTATION

The algorithms described here have been implemented in a simple molecular ray-tracing program called MolRay, written in standard C. This program currently runs on PC-compatible machines and small Unix workstations, such as the Sun SPARC-station and the Silicon Graphics Personal Iris. MolRay is modular in design, comprising a core-rendering module capable of ray tracing primitive objects (currently spheres and triangles), and preprocessing modules that generate files of primitives from protein data files. The examples shown here were generated with the *balls* module (for CPK models), and the *worms* module (for smoothed α-carbon trace models). Both modules take as input a file of atomic coordinates in the Brookhaven Protein Data Bank (PDB) format and a script file that controls the view specifications and the object characteristics of atoms (such as radius, color, gloss, reflectivity, and so on). The script file uses Unix-style regular expression matching to process the PDB file. The following snippet of a MolRay script will result in the α-carbon atoms of all proline residues being displayed as glossy red spheres:

```
IF ^ATOM.....CA..PRO
COLOUR 0.9 0.0 0.0
GLOSS 150.0
REFLECT 0.0
```

Combinations of such regular expression clauses can provide complete rendering dictionaries for different applications. See the accompanying Color Plates for examples.

Up to five light sources are permitted, with three lighting models: point (uniformly radiating point source), diffuse (uniformly radiating scattered light source), and spot (directional light source). The spot model implements a subset of the controls described by Warn,[9] providing photographic spotlight effects with variable focus and aperture. Spotlight effects are useful for drawing attention to important aspects of a molecular model, such as binding sites.

In all cases the images shown in Color Plates 1–5 were generated by calculating four rays per pixel to reduce the effects of aliasing. Color Plate 1 shows an example of a ray-traced image rendered on a microcomputer. Two light sources were used (plus an ambient light source): a point source in front of the model, and a highly directional spotlight shining across the middle of the model from the right. This display was rendered at a resolution (more correctly an addressability) of 320×200 (8 bitplanes), which on a 20-MHz 386-PC, with a 387-coprocessor and VGA graphics card, was completed in 30 minutes. The troponin-*C* model shown in Color Plate 2 consists of 1127 atoms. Two light sources were used: a diffuse source in front and a point source above and to the right. The image resolution in this case was 640×400 (8 bitplanes), and took 168 minutes to render on the same 386-PC system.

An implementation of MolRay for the Personal Iris (4D/20) was used to generate Color Plates 3–5. Color Plate 3 shows the same CPK model of troponin-*C*, rendered at a resolution of 1280×1024 (24 bitplanes) on the Iris in 133 CPU minutes. A worm display of the troponin-*C* structure is shown in Color Plate 4, which is constructed by drawing 7500 spheres at *B*-spline-interpolated positions along the α-carbon trace. Color Plate 4 was rendered at a resolution of 1280×1024 (24 bitplanes), in this case taking 158 CPU minutes. Color Plate 5 shows the relationship between the two subunits (the *A*-chain, colored white, and the *B*-chain, colored red) of bovine catalase (Brookhaven code 8CAT). This CPK model comprises 8016 atoms, and took 260 CPU minutes to render on the Iris at a resolution of 1280×1024 (a 24 bitplane display was used, though only 8 were required).

Output from MolRay is produced in the form of an optional raw RGB pixel file, which contains ideal 24-bit RGB values for each generated pixel and a screen display (which may be saved separately). For 24-bit frame buffers, there is no difficulty in displaying the image on screen, but for 8-bit frame buffers compromises must be made. To avoid slowing down the rendering process, no attempt is made to dither the display; instead an attempt is made to pick the best palette of 256 colors before starting the ray trace and to select the closest of these for each rendered pixel color. The palette choice is made by considering the range of colors

defined in the script file, and the additional shades likely to be required after lighting and shadowing are taken into account. To make the most of the limited number of colors available, the base color choice in the script is limited to 4 RGB levels, giving a choice of 64 colors. This limit is not applied when 24-bitplane images are generated, in which case base atom colors can be picked from any of the available colors.

CONCLUSION

Ray tracing is a very powerful means for generating realistic molecular model displays. Though it is computer-intensive, the methods described here demonstrate how shortcuts can be made in the process, enabling reasonable displays to be generated with only modest computing resources. Though the examples shown here are based solely on sphere primitives, these methods are easily extended to the rendering of triangles, boxes, cylinders, disks, and even more complex three-dimensional shapes. Using this extended set of primitives, extensions to MolRay are being developed to allow the generation of other forms of molecular display, such as Richardson protein secondary structure diagrams.

REFERENCES

1 Palmer, T.C., Hausheer, F.H. and Saxe, J.D. Applications of ray tracing in molecular graphics. *J. Mol. Graphics* (1989) **7**, 160–164

2 Lauher, J.W. Chem-Ray: A molecular graphics program featuring an umbra and penumbra shadowing routine. *J. Mol. Graphics* (1990) **8**, 34

3 Appel, A. Some techniques for shading machine renderings of solids. *AFIPS 1968 Spring Joint Comp. Conf.* (1968) **32**, 37–45

4 Whitted, T. An improved illumination model for shaded display. *Comm. ACM* (1980) **23**, no. 6, 343–349

5 *An Introduction to Ray Tracing*. (Glassner, A.S., Ed.) Academic Press, New York, 1989

6 Williams, L. Casting curved shadows on curved surfaces. *Comp. Graphics* (1978) **12**, 270–274

7 Haines, E. Essential ray tracing algorithms. In: *An Introduction to Ray Tracing* (A.S. Glassner, Ed.) Academic Press, New York, 1989, 33–77

8 Zupan, J. *Clustering of Large Data Sets*. Research Studies Press, Chichester, 1982

9 Warn, D.R. Lighting controls for synthetic images. *Comp. Graphics* (1983) **17**, 13–21

10 Wlodawer, A., Walter, J., Huber, R. and Sjolin, L. Structure of bovine pancreatic trypsin inhibitor: Results of joint neutron and X-ray refinement of crystal form II. *J. Mol. Biol.* (1984) **180**, 301

11 Satyshur, K.A., Rao, S.T., Pyzalska, D., Drendel, W., Greaser, M. and Sundaralingam, M. Refined structure of chicken skeletal muscle troponin C in the two-calcium state at 2-Å resolution. *J. Biol. Chem.* (1988) **263**, 1628

12 Fita, I., Silva, A.M., Murthy, M.H.W., and Rossmann, M.G. The refined structure of beef liver catalase at 2.5-Å resolution. *Acta Crystallogr., Sect. B* (1986) **42**, 497